Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K”

Виконав здобувач вищої освіти
IV курсу, групи КБ-19
ОПП «Кібербезпека»
спеціальності 125 «Кібербезпека»
_____ Степаненко Є.О.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук, доцент
_____ Смірнов С.А.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет

Факультет *Механіко-технологічний*

Кафедра *Кібербезпеки та програмного забезпечення*

Освітній ступінь *бакалавр*

Галузь знань ._12 _____*“*Інформаційні технології*”*_____

Спеціальність *125 “*Кібербезпека*”*

Освітньо-професійна (освітньо-наукова) програма _*“*Кібербезпека*”*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Степаненку Євгенію Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи *Програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K*

2. Керівник роботи *Смірнов Сергій Анатолійович, канд. техн. наук, доцент*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 12-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.

2.Перегляд аналогічних існуючих систем.

3. Опис і обґрунтування проектних рішень.

4. Етапи програмування системи.

5. Впровадження системи кібербезпеки в промислову експлуатацію.

6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

Структурна схема системи кібербезпеки	*1 аркуш*
Функціональна схема системи кібербезпеки	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Смірнов С.А.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Степаненко Є.О.
(прізвище та ініціали)

АНОТАЦІЯ

Степаненко Є.О. Програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K. 125 Кібербезпека. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

Метою розробки є програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

Результат роботи – програмна реалізація системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Delphi 10.

Ключові слова: кібербезпека, Intel Core i9-10900K

ABSTRACT

Stepanenko E.O. Cybersecurity system software to ensure information protection using special instructions Intel Core i9-10900K. 125 Cyber security. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this final qualification work for the first (bachelor) level of higher education, software is developed, which is intended for a cyber security system to ensure information protection using special Intel Core i9-10900K instructions.

The purpose of the development is the software of the cyber security system to ensure the protection of information using the special instructions of the Intel Core i9-10900K.

The result of the work is the software implementation of the cyber security system to ensure information protection using special Intel Core i9-10900K instructions.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Delphi 10 environment.

Keywords: cybersecurity, Intel Core i9-10900K

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

КСЗ	–	комплексна система захисту
ПЕОМ	–	персональна електронно-обчислювальна машина
СУБД	–	система управління базами даних
PGP	–	Pretty Good Privacy

ВСТУП

Актуальність теми. Механізм Intel Advanced Encryption Standard (AES) або New Instructions (AES-NI) забезпечує високошвидкісне апаратне шифрування та дешифрування для OpenSSL, ssh, VPN, повного дискового шифрування Linux/Unix/OSX тощо.

Advanced Encryption Standard Instruction Set і Intel Advanced Encryption Standard New Instructions дозволяють певним процесорам Intel/AMD та іншим процесорам виконувати надзвичайно швидке апаратне шифрування та дешифрування.

Зауважте, що підтримка AES-NI вмикається автоматично, якщо виявлений процесор входить до списку підтримуваних, як зазначено вище. Список процесорів, які підтримують механізм AES-NI, див. на сайті Intel ARK/AMD/ARM (постачальник)/VIA і в документації. AES-NI є розширенням архітектури набору інструкцій x86 для мікропроцесорів Intel і AMD. Це збільшує швидкість програм, які виконують шифрування та дешифрування за допомогою AES. Кілька постачальників серверів і ноутбуків постачали конфігурації BIOS з вимкненим розширенням AES-NI.

Підтримуються такі ЦП:

1. Intel Westmere/Westmere-EP (Xeon 56xx)/Clarkdale (крім Core i3, Pentium і Celeron)/Arrandale (крім Celeron, Pentium, Core i3, Core i5-4XXM).
2. Процесор Intel Sandy Bridge (крім Pentium, Celeron, Core i3).
3. Intel Mobile Core i7 і Core i5.
4. Процесори Intel Ivy Bridge Лише всі i5, i7, Xeon і i3-2115C.
5. Процесори Intel Haswell (усі, крім i3-4000m, Pentium і Celeron).
6. Intel Coffee Lake/Kaby Lake тощо
7. Процесори на базі AMD Bulldozer/Piledriver/Steamroller/Jaguar/Puma/Ryzen.

8. Процесори AMD Geode LX.

9. VIA PadLock (набір інструкцій, відмінний від Intel AES-NI, але виконує те саме в кінці дня).

10. ARM – обрано Allwinner і Broadcom за допомогою процесора безпеки. Існує ще кілька процесорів на основі ARM.

11. Багато останніх процесорів Intel підтримують AES-NI (розширене шифрування) і ввімкнено за допомогою параметра BIO.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

– Дослідження системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

– Програмна реалізація системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Сьогодні безпека є важливою темою – однак важливою її вважають, головним чином, тільки професіонали. Втім, якщо безпека стає маркетинговим елементом або перетворюється в характеристику продуктивності, то такі компанії, як Intel, починають активно неї просувати. Чи вважаєте ви чи ні, що на вашій системі є конфіденційні дані, залежить від того, що ви маєте на увазі під цими даними, а також і від вашого персонального рівня комфорту. Крім того, безпека завжди має на увазі правильну стратегію й акуратність у зберіганні конфіденційних даних. Ніколи не можна залишати без уваги такі дані, як реквізити вашого паспорта або номер і дату закінчення терміну дії банківської карти. Або навіть PIN-код телефону.

Одне можна сказати точно: найкраще бути акуратним і розсудливим, ніж навпаки – тим більше що для цього потрібно не так багато зусиль. Підхід Intel до додавання прискорення AES не охоплює всі додатки шифрування й сценарії, тільки самий популярний стандарт – при цьому ви одержите все це безкоштовно у всіх майбутніх 32-нм настільних процесорах для масового ринку або для більше дорогих сегментів.

AES розшифровується як "Advanced Encryption Standard" – це найбільш популярний стандарт симетричного шифрування у світі IT. Стандарт працює із блоками розміром 128 біт і підтримує 128-, 192– або 256-бітні ключі (AES-128, AES-192 і AES-256). Багато утиліт шифрування, та ж TrueCrypt, підтримали алгоритм AES на самому початку його існування. Але найбільший фактор успіху AES, звичайно, полягає в його прийнятті урядом США в 2002 році, при цьому в 2003 році він був прийнятий як стандарт для захисту секретних даних.

Шифрування даних за допомогою AES

Шифрування AES базується на системі підстановок з перестановкою, тобто над даними проводиться серія математичних операцій, щоб створити значно модифікований масив даних (зашифрований). Як вихідна інформація виступає текст, а ключ відповідає за виконання математичних операцій. Операції можуть бути як зовсім простими, наприклад, зрушення біт або XOR, так і більш складними. Один прохід можна легко розшифрувати, тому всі сучасні алгоритми шифрування побудовані на декількох проходах. У випадку AES це 10, 12 або 14 проходів для AES-128, AES-192 або AES-256. До речі, ключі AES проходять таку ж процедуру, що й користувальницькі дані, тобто вони являють собою що змінюється раундовий ключ.

Процес працює з масивами 4x4 з одиночних байтів, також називаних боксами: S-box використовуються для підстановок, P-box – для перестановок. Підстановки й перестановки виконуються на різних етапах: підстановки працюють усередині так званих боксів, а перестановки міняють інформацію між боксами. S-box працює по складному принципу, тобто навіть якщо єдиний вхідний біт буде мінятися, то це вплине на декілька вихідних біт, тобто властивості кожного вихідного біта залежать від кожного вхідного біта.

Використання декількох проходів забезпечує гарний рівень шифрування, при цьому необхідно відповідати критеріям розсіювання (diffusion) і заплутування (confusion). Розсіювання виконується через каскадну комбінацію трансформацій S-box і P-box: при зміні тільки одного біта у вхідному тексті S-box буде модифікувати вихід декількох біт, а P-box буде псевдовипадково поширювати цей ефект по декількох S-box. Коли ми говоримо про те, що мінімальна зміна на вході дає максимальну зміна на виході, ми говоримо про ефект сніжної грудки.

Останнім часом іде чимало обговорень так званих взломів, які обходять необхідність запуску розширеного пошуку методом грубої сили для знаходження правильного ключа розшифровки. Технології, такі як атаки XSL і атаки related-

key обговорюються досить інтенсивно – але успіх невеликий. Єдиний працюючий спосіб злому шифрування AES полягає в так званій атаці побічного каналу (side-channel). Для її здійснення атака повинна відбуватися тільки на host-системі, на якій виконується шифрування AES, і при цьому вам необхідно знайти спосіб одержання інформації про синхронізацію кеша. У такому випадку можна відстежити число тактів комп'ютера до завершення процесу шифрування.

Звичайно, все це не так легко, оскільки вам потрібен доступ до комп'ютера, причому досить повний доступ для аналізу шифрування й права на виконання коду. Тепер вам напевно зрозуміло, чому "діри" у системі безпеки, які дозволяють зловмисникові одержати такі права, нехай навіть вони звучать зовсім абсурдно, необхідно закривати якнайшвидше. Якщо ви одержите доступ до цільового комп'ютера, то добування ключа AES – справа часу, тобто вже не трудомістке завдання для суперкомп'ютерів, що вимагає величезних обчислювальних ресурсів.

1.2 Область застосування

На даний момент інтегровані в CPU інструкції AES починають мати сенс – незалежно від можливих переваг по продуктивності. З погляду безпеки процесор може обробляти інструкції AES в інкапсульованому виді, тобто йому не потрібні які-небудь таблиці перетворення, необхідні для атаки методом побічного каналу.

Core i9-10900K Clarkdale з підтримкою AES

Процесори насправді знаменують собою зміну поколінь, оскільки при цьому не тільки відбувається перехід на наступний техпроцес (32 нм у порівнянні з 45 нм), але перед нами й перше покоління CPU з підтримкою декількох інструкцій, що прискорюють шифрування. Intel згадує добавку як AES New Instructions. Вони складаються із:

– інструкцій для шифрування AES (AESENC, AESENCLAST);

– інструкцій для розшифровки (AESDEC, AESDECLAST);

– інструкції для роботи із ключем AES (AESIMC, AESKEYGENASSIST).

Як і раніше, інструкції відносяться до SIMD, тобто до типу "одна інструкція багато даних" (Single Instruction Multiple Data). Підтримуються всі три ключі AES (128, 192 і 256 біт з 10, 12 і 14 проходами підстановки й перестановки).

Оскільки всі інструкції AES мають фіксовану затримку, що не залежить від даних, тобто час фіксований й доступ до пам'яті не потрібно. Крім того, модель програмування така ж, як і у випадку інших інструкцій SSE з первісного стандарту SSE4. Таким чином, всі операційні системи, які підтримують роботу з SSE, зможуть використовувати й інструкції AES New Instructions.

Будьте обережні при виборі процесора із прискоренням AES, оскільки сьогодні лише деякі моделі підтримують нові інструкції. 32-нм процесори Core i3 на Clarkdale не підтримують інструкції, а двоядерна лінійка Core i9-10900K-600 – підтримує. У мобільних процесорів ситуація ледве більш складна: якщо мобільні процесори Core i3 теж не підтримують прискорення AES, то процесори лінійки Core i9-10900K-500 уже підтримують. Однак є одна модель Core i9-10900K-400, позбавлена такої підтримки. Усе було б набагато простіше, якби Intel додала підтримку нових інструкцій в усі моделі.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

У рамках виконання бакалаврської роботи проведемо тестування ряду програмних продуктів на Core i9-10900K Clarkdale з підтримкою AES.

Результати тестів

SiSoftware Sandra SP3

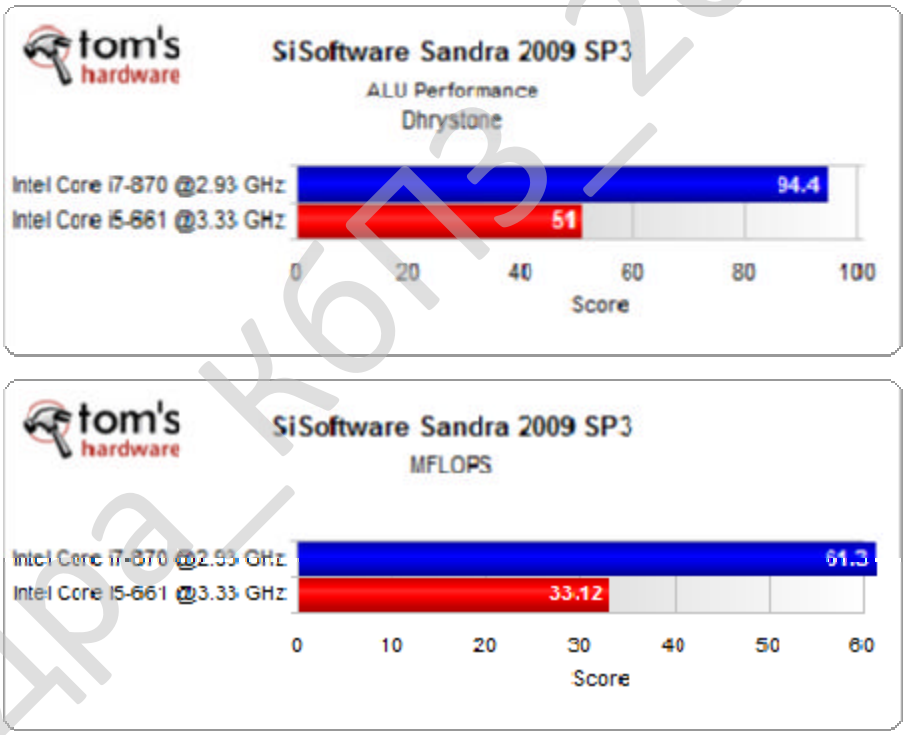

Рисунок 2.1 – Тести ALU і MFLOPS

Тести ALU і MFLOPS не дали яких або сюрпризів: чотирьохядерний процесор майже у два рази швидше, незважаючи на меншу тактову частоту – як ми й очікували.

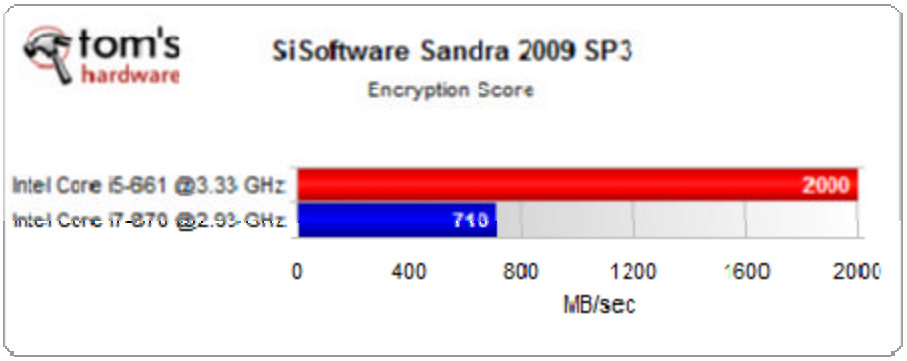

Рисунок 2.2 – Тест шифрування AES

Втім, результат тесту шифрування зовсім інший: він говорить про те, що процесор Core i9-10900K Clarkdale із прискоренням AES до 3x швидше чотирьохядерного Core i 7-870.

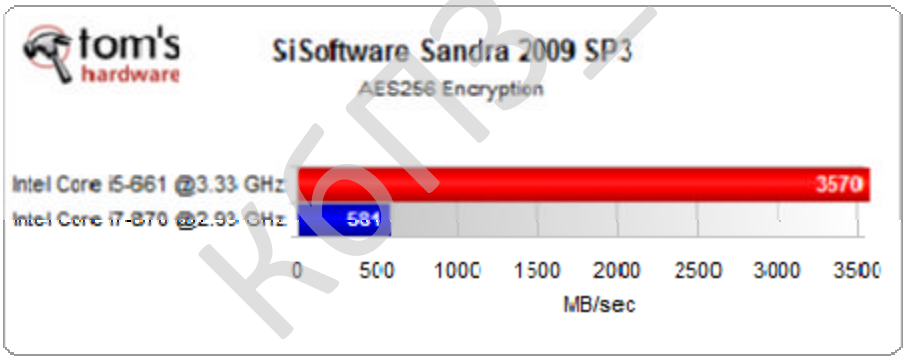

Рисунок 2.3 – Тест шифрування AES-256

Саме тому результати тесту шифрування виглядають так добре: чисте шифрування AES-256 виконується більш ніж у шість разів швидше на двоядерному процесорі з апаратним прискоренням.

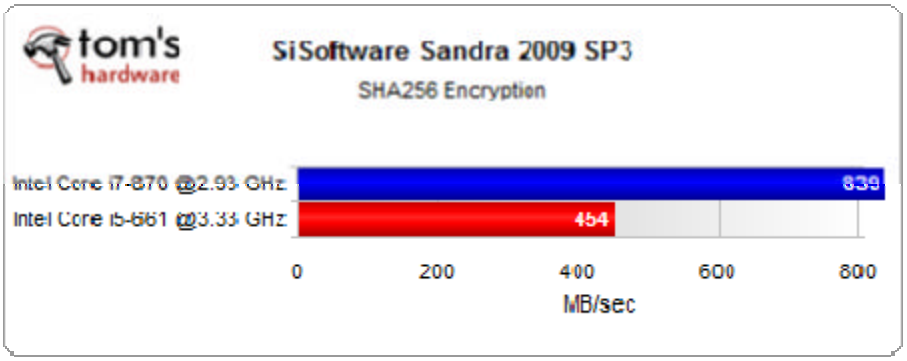

Рисунок 2.4 – Тест шифрування SHA-256

Тест шифрування SHA-256 доводить, що дана функція прискорює тільки алгоритм AES.

PCMark Vantage Communications Test

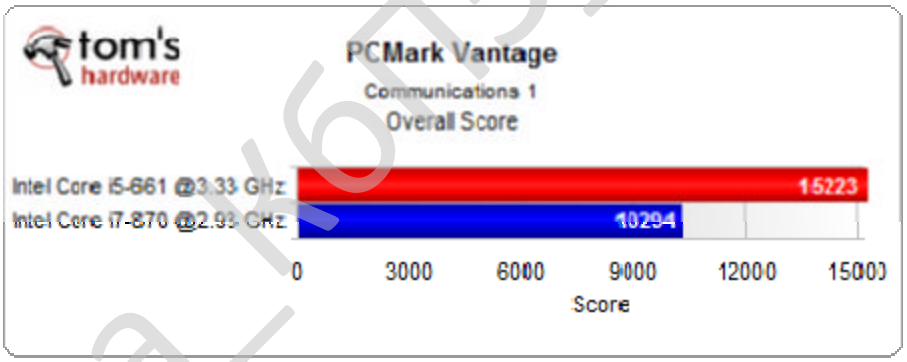

Рисунок 2.5 – Тест пакета Communications

Тест PCMark Vantage повідомляє нас рівно той же саме: загальний результат пакета Communications на 50% швидше на новому 32-нм двоядерному процесорі Clarkdale у порівнянні з 45-нм чотирьохядерним Lynnfield.

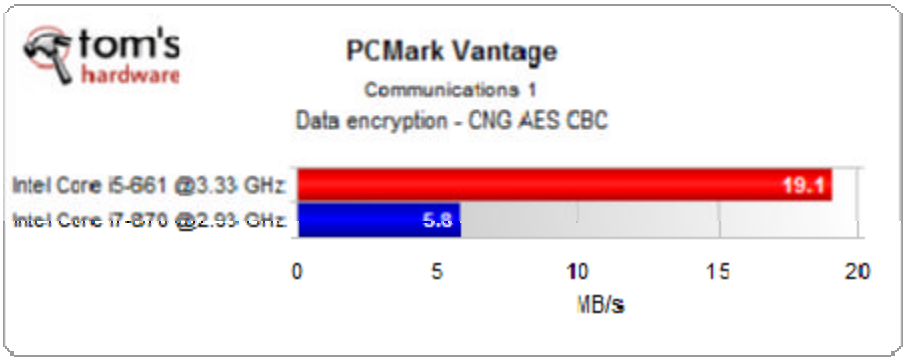

Рисунок 2.6 – Тест пакета Communications (шифрування AES)

А от і причина.

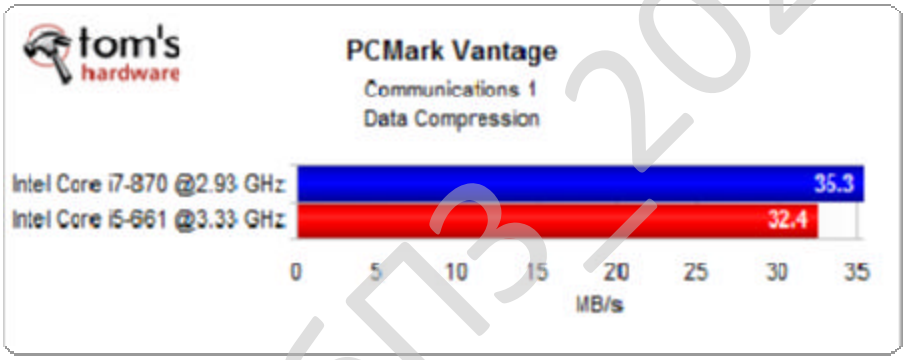

Рисунок 2.7 – Тест пакета Communications (стиск)

Знову ж: Intel прискорила тільки алгоритм AES. Стиск даних від цього не виграє, швидкість залежить від числа ядер і тактової частоти.

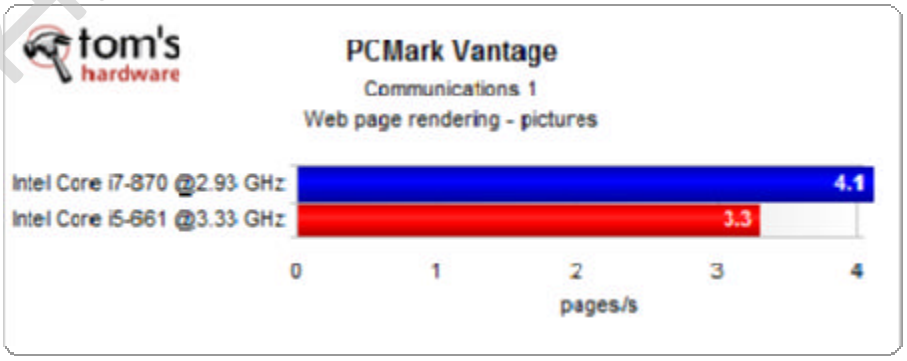

Рисунок 2.8 – Тест пакета Communications (рендеринг)

Розшифровка алгоритму AES теж досить сильно прискорюється.

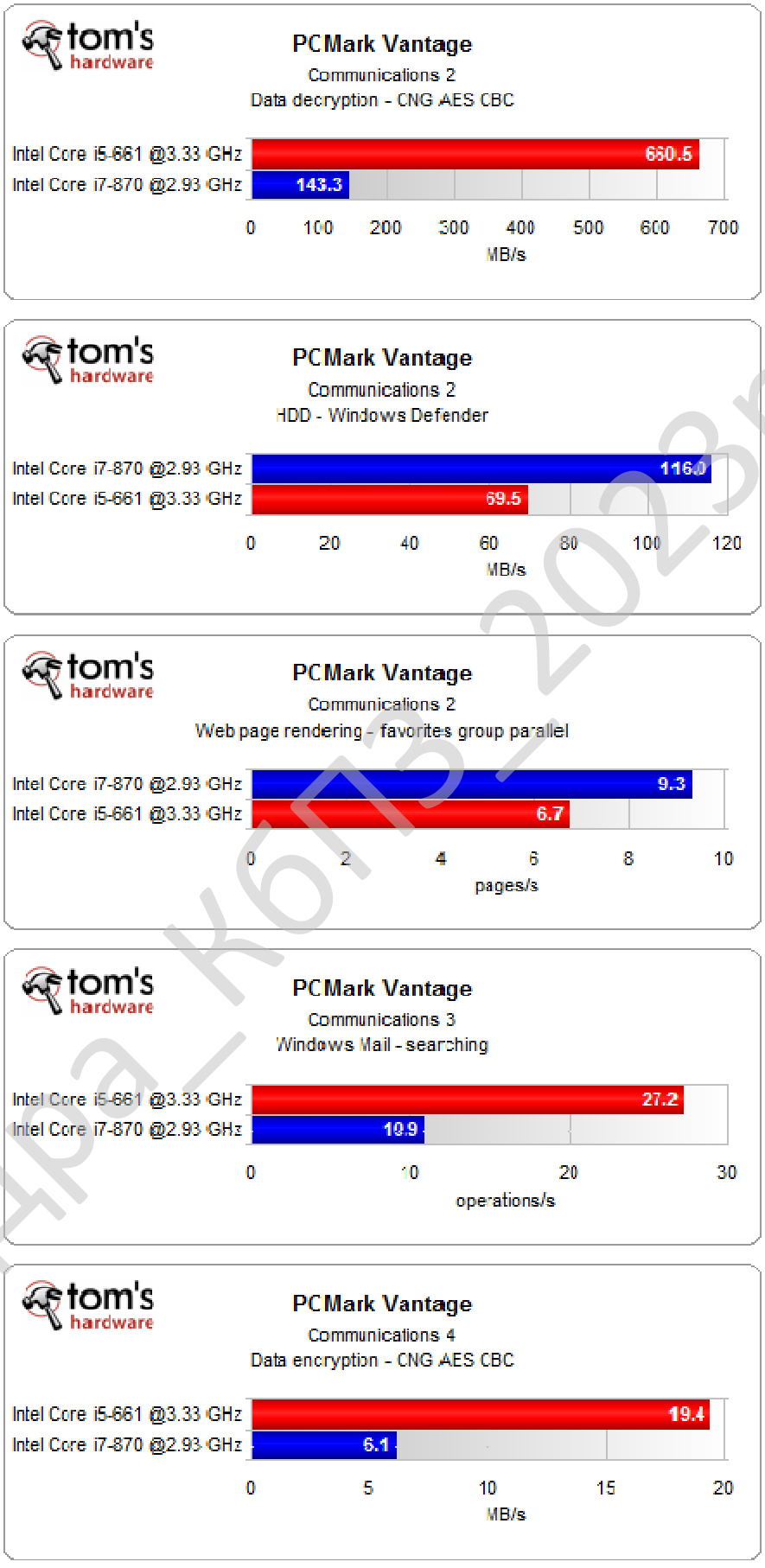

Рисунок 2.9 – Тести пакета Communications

Перед вами ще один тест, у якому двоядерний Clarkdale Core i9-10900K добре показує себе завдяки підтримці AES.

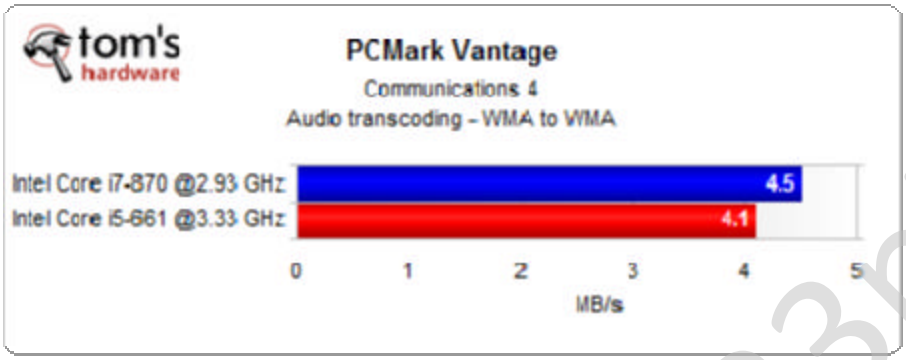

Рисунок 2.10 – Тест пакета Communications (перекодування аудіо)

Bitlocker, Everest і WinZIP 14

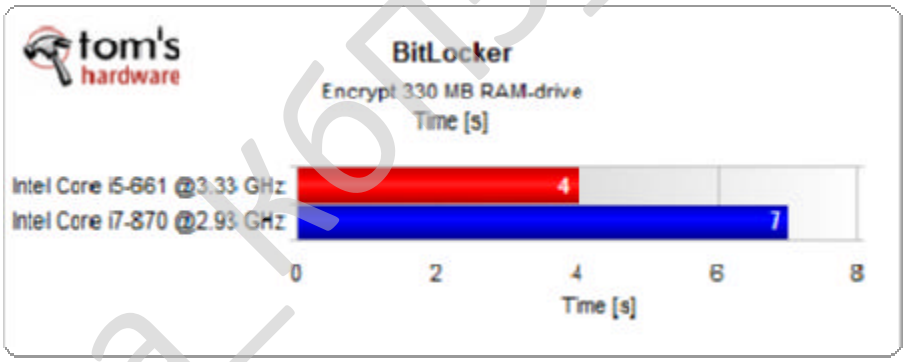

Рисунок 2.11 – Тест BitLocker

Ми використовували функцію шифрування диска BitLocker, вбудовану в операційну систему Windows 11 Ultimate, що теж базується на AES. Щоб не впертися в "вузьке місце" у вигляді накопичувача, ми вирішили використовувати 330-Мбайт RAM-диск, що зможе показати різницю в продуктивності між двоядерним Core i9-10900K-661 із прискоренням AES проти чотирьохядерного Core i 7-870, що не підтримує прискорення шифрування. Фактично, різниця виявилася близька до 50%, шифрування зайняло 7 секунд на потужному

чотирьохядерному Core i7, а новий 32-нм двоядерний Core i9-10900K-661 упорався з тим же самим завданням усього за 4 секунди.

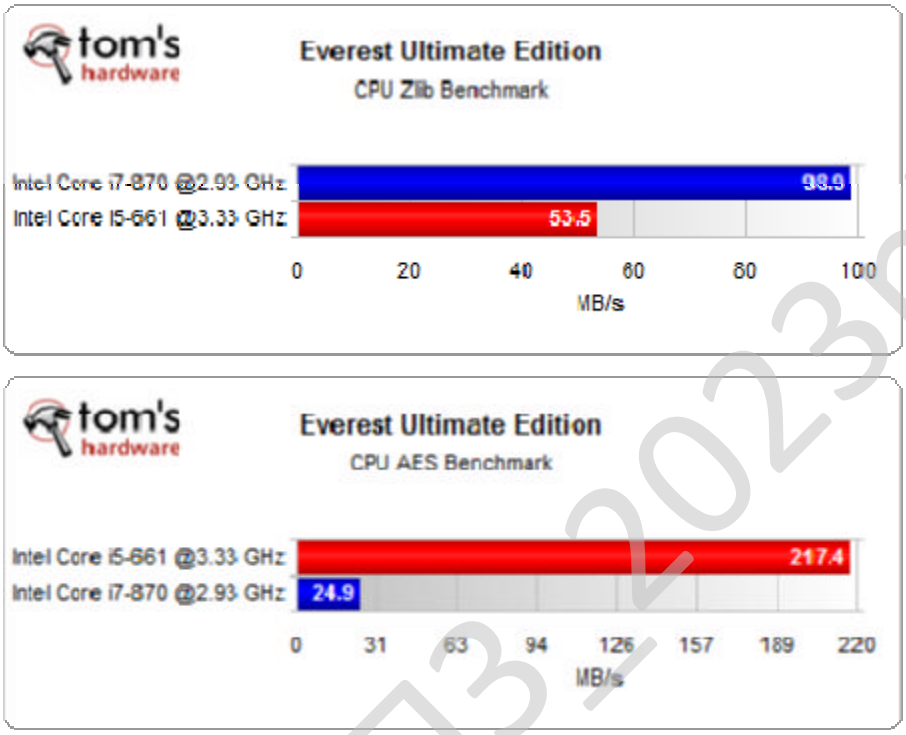

Рисунок 2.12 – Тест Everest Ultimate Edition

Тест шифрування AES у пакеті Everest Ultimate Edition демонструє фантастичний приріст продуктивності, втім, це однаково залишається більше теоретичним результатом.

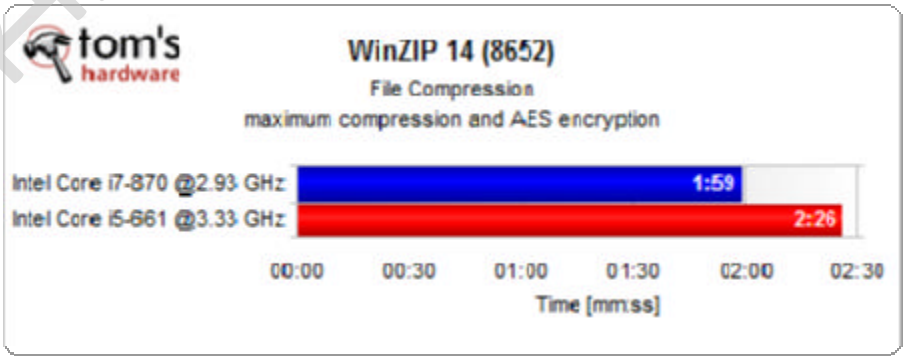

Рисунок 2.13 – Тест WinZIP 14

Незважаючи на підтримку AES, тест WinZIP 14 виконувався швидше на чотирьохядерному процесорі через більшу потужність. Втім, двоядерний процесор з апаратним прискоренням AES усе ще показав себе добре, програвши тільки з тієї причини, що ми вказали максимальний рівень стиску. Ми вибрали подібний режим, оскільки більшість користувачів виберуть саме його, якщо тільки немає яких-небудь причин зменшити рівень стиску (наприклад, щоб швидше запакувати великий масив даних).

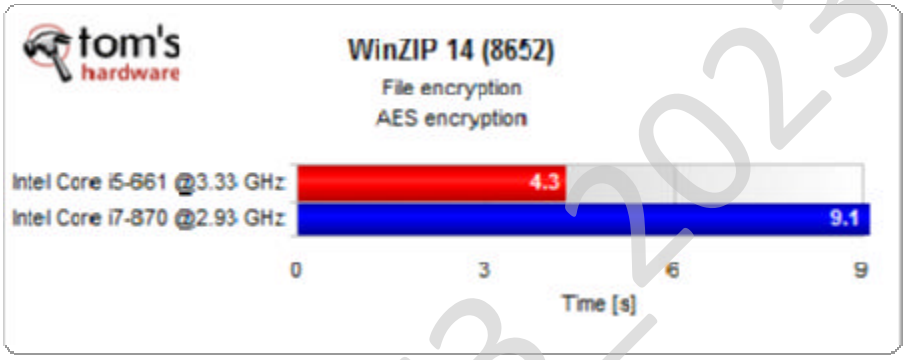

Рисунок 2.14 – Тест із шифруванням AES в WinZIP

Ми повторили цей тест із шифруванням AES в WinZIP, але рівень стиску був нульовим – тобто WinZIP просто переписував файли в архів. І тут ми бачимо, що двоядерний процесор із прискоренням AES дійсно обганяє чотирьохядерну модель, де такої функції немає.

7-zip

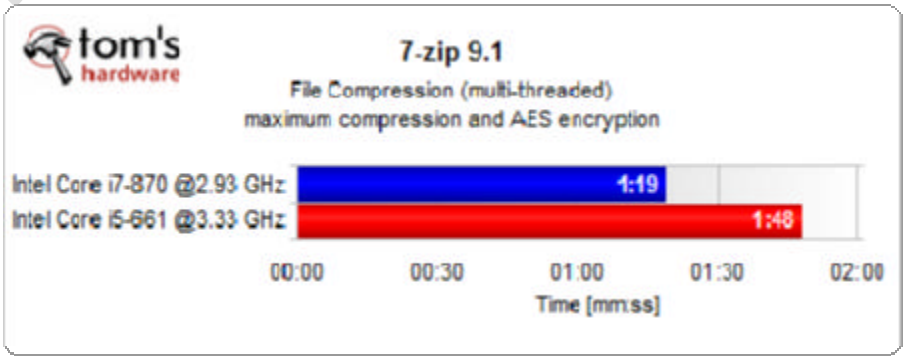

Рисунок 2.15 – Тест 7-zip

Ми використовуємо тест стиску файлів, що запускається з командного рядка, і знову бачимо, що чотирьохядерний процесор працює швидше. Знову ж, причина криється у високому рівні стиску. Ми повторили тест із нульовим рівнем стиску файлів.

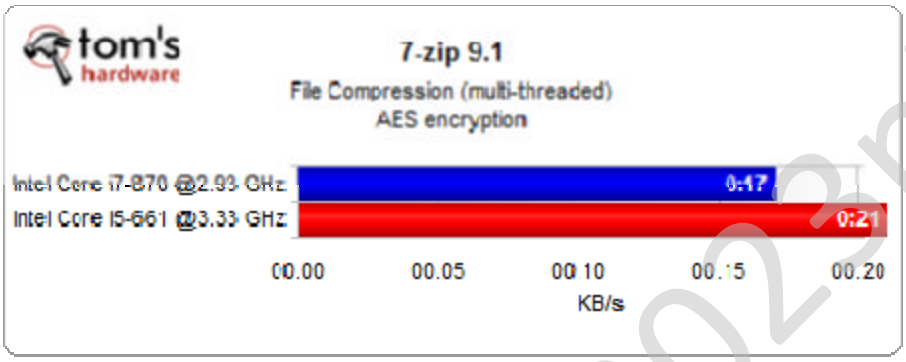

Рисунок 2.16 – Тест 7-zip із шифруванням AES

У цьому випадку ми бачимо мало користі від Core i9-10900K-661 і убудованої функції AES. Час додавання файлів в архів становить одну п'яту частину від часу при високому рівні стиску, але чотирьохядерний процесор однаково виходить уперед.

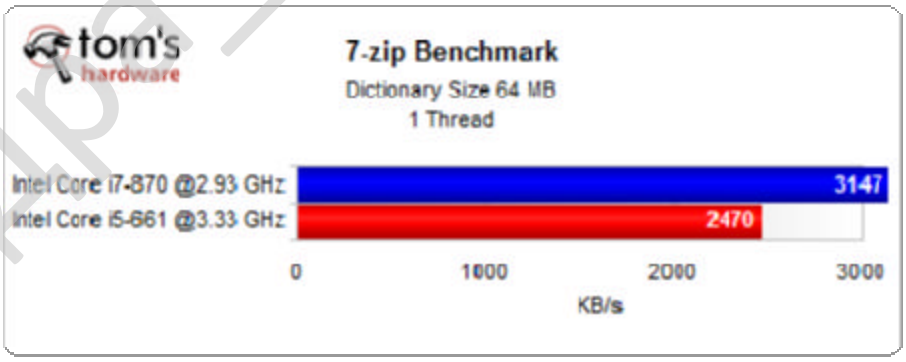

Рисунок 2.17 – Тест 7-zip 1 потік

Як ви можете бачити, на наступних діаграмах, більше число потоків приводять до збільшення продуктивності 7-zip.

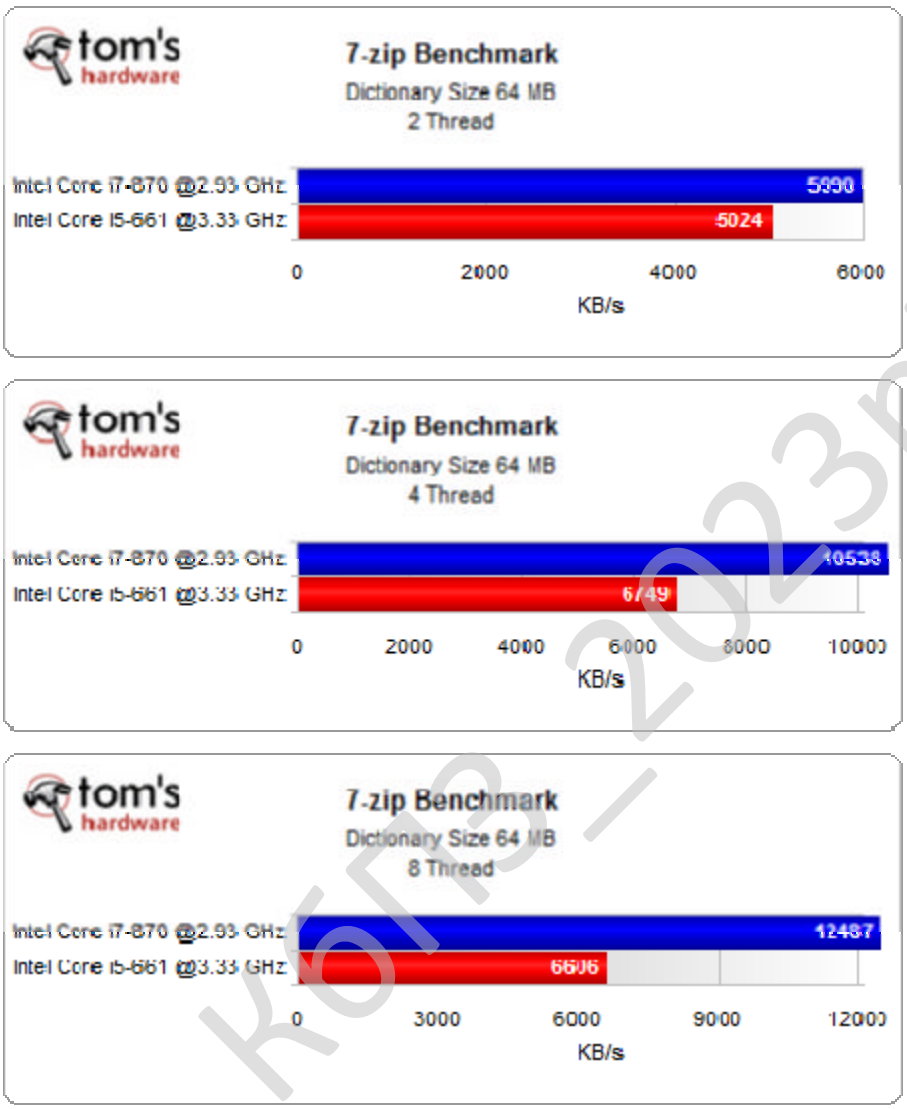

Рисунок 2.18 – Тест 7-zip різне число потоків

Висновок

Наш аналіз дав кілька цікавих результатів, але також і показав, що не всі додатки відразу ж можуть виграти від шести нових інструкцій Intel AES New Instructions, які були розроблені для прискорення шифрування й розшифровки алгоритмів AES-128, AES-192 і AES-256. Результати тестів показали досить серйозну продуктивність двоядерного Core i9-10900K-661 Clarkdale, що ми порівнювали із чотирьохядерним Core i 7-870. Тести PCMark Vantage і SiSoftware

Sandra показали вражаючий приріст продуктивності через апаратне прискорення AES. Результати Everest Ultimate дуже схожі.

Втім, всі згадані тестові пакети є синтетичними, тобто вони звичайно демонструють більшу різницю, ніж можна побачити у звичайному житті. Із цієї причини ми також провели тести архіватора 7-zip 9.1 Beta, BitLocker під Windows 11 Ultimate і WinZIP в останній версії 14, щоб перевірити, які переваги ми одержимо в реальних додатках. Треба сказати, ми не розчарувалися: WinZIP 14 і Bitlocker давали результати, які практично повторили виграш, отриманий у синтетичних тестових пакетах.

Але тест архіватора 7-zip 9.1 beta, що повинен підтримувати нові інструкції прискорення AES, не показав помітної переваги – або воно було занадто малим, поступившись високої обчислювальної потужності чотирьохядерного Core і 7-870, що працює на частоті 2,93 ГГц, що досить близько до номінальної тактової частоти 3,33 ГГц в Core i9-10900K-661.

У підсумку ми можемо підтвердити, що підхід Intel дійсно себе виправдав, хоча не можна сказати, що все програмне забезпечення, що використовує AES, прискорюється. Втім, переваги по більш надійному захисту залишаються, оскільки апаратне прискорення шифрування й розшифровки AES запобігає можливість атаки методом побічного каналу, коли ключ AES витягає шляхом відстеження ділянок доступу до пам'яті (кеша).

2.2 Обґрунтування вибору засобів для побудови системи кібербезпеки та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, Mac OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в

чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium, використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4 забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні 64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.

– Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи кібербезпеки контролю роботи технологічного обладнання на виробництві в автоматизованому

режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи кібербезпеки в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Як вийшло, що для настільних систем компанія Intel досі не змогла запропонувати нічого принципово нового, писалося вже не один раз. Виведення нових мікроархітектур у мікропроцесорного гіганта було завжди прив'язане до змін виробничих норм, а після запуску 14-нм технології з подальшими техпроцесами справа не залагодилася. Виробничий процес з роздільною здатністю 10 нм, згідно з початковим планом, повинен був бути освоєний ще в далекому 2017 році, і, якби все склалося, як замислювалося, зараз у ході були б зовсім інші процесори. Однак впровадження 10-нм норм обернулося для Intel справжньою катастрофою, і навіть зараз, коли йдеться про запуск вже другої версії цієї проблемної технології, використовувати її при випуску великих та високочастотних напівпровідникових кристалів компанія все ще не наважується.

Тим не менш, як би нам не набридли чергові втілення Skylake в кремнії, не можна заперечувати, що це безумовно вдале рішення, яке має право на таке довгожительство. Сумніви у її актуальності можуть бути лише у світлі використання техпроцесу з великими допусками, але серйозні претензії до неї пред'явити непросто. По-перше, незважаючи на щорічну появу нових ітерацій AMD Ryzen, процесори з мікроархітектурою Skylake все ще виглядають цілком конкурентоспроможно за питомою продуктивністю на такт, тобто за показником IPC. По-друге, Skylake виявилася мікроархітектурою, що легко масштабується. Не змінюючи нічого в принципах її внутрішнього пристрою, Intel зуміла в два з половиною рази наростити кількість процесорних ядер і не втратити при цьому ефективності їх взаємодії. Більш того, дуже непогано виглядають і тактові частоти, що досягаються сучасними процесорами Intel. Актуальні 14-нм послідовники Skylake впевнено подолали позначку в 5

ГГц, тоді як конкуруючим рішенням, що випускаються за 7-нм техпроцесом, таких вершин досягти так і не вдалося.

До того ж, сьогодні ми журимося з приводу використання в нових процесорах Intel архітектури п'ятирічної давності, ймовірно, востаннє. Послідовники Comet Lake-S для масового сегменту зможуть відмовитись від цієї спадщини вже досить скоро. У наступному поколінні чіпів, що фігурує під кодовим ім'ям Rocket Lake-S, компанія Intel планує використати прогресивну мікроархітектуру Willow Cove з мобільних процесорів Tiger Lake, перекладену на 14-нм техпроцес. Принципове оновлення мікроархітектури попутно має відбутися і в сегменті HEDT, де за кілька місяців з'являться процесори з дизайном Ice Lake-X, що виробляються за 10-нм технологією.

Але не забігатимемо вперед: поки ми маємо жити ще з однією реінкарнацією Skylake. І головне питання, яке стоїть перед сьогоднішнім тестуванням, полягає в тому, чи зможе Intel, спираючись виключно на старий технологічний фундамент, дати відповідь на процесори Ryzen третього покоління, які досить агресивно стали тіснити Coffee Lake-S на всіх фронтах. Відстоювати свої позиції мікропроцесорний гігант планує дуже простими засобами: додаванням флагманським процесорам кількох обчислювальних ядер, а в процесорах середньої та молодшої ланки – включенням технології Hyper-Threading.

Що нового в Comet Lake-S

Здавалося б, ми знаємо Skylake вздовж і поперек, але Intel знову знайшла, як можна покращити продукти минулого покоління, не вдаючись ні до нових виробничих технологій, ні до змін у мікроархітектурі. І головне додавання такого роду – це два додаткові обчислювальні ядра, які отримали старші моделі процесорів Core i9, що належать до сімейства Comet Lake-S. Таким чином, тепер масова платформа Intel може бути укомплектована процесором, що має відразу десять обчислювальних ядр.

Пару додаткових ядер додали до Comet Lake-S за вже відпрацьованою

схемою. Їх фактично приєднали до наявного кристала збоку, підключивши все до тієї ж кільцевої шини, яка, до речі, здатна «витягнути» і 12 ядер (такі конфігурації використовувалися в серверних процесорах Broadwell-E). Природно, додаткові ядра укомплектовані і встановленими ним сегментами кеш-пам'яті третього рівня об'ємом по 2 Мбайт. Відповідно, десятиядерник, що вийшов, оснащений в цілому 20 Мбайт L3-кеша.

Все це добре видно по знімку напівпровідникового кристала 10-ядерної версії Comet Lake-S: він дуже схожий на Coffee Lake-S, а різниця лише в кількості ядер.

Зрештою площа 10-ядерного кристала Comet Lake-S становить близько 198 мм ^{2} і це на 14 % (очікувано) більше площі восьмиядерного кристала Coffee Lake-S. Для порівняння варто нагадати, що восьмиядерний 7-нм чіплет CCD процесорів Zen 2 має площу всього 74 мм ^{2} але площа 12-нм I/O-чіплета досягає 125 мм ^{2}. Тобто, як це не дивно, за сумарними розмірами кремнію між 10-ядерним Comet Lake-S і 8-ядерним Zen 2 можна поставити знак зразкової рівності.

Об'єднання з десятка ядер, виконаних по 14-нм техпроцесу, важко зробити економічною, не вдаючись до суттєвого урізання тактових частот. Тому старші процесори покоління Comet Lake-S помітно наростили свої енергетичні апетити та, відповідно, тепловиділення. Рамки теплового пакета, заявлені в специфікаціях, відсунулися з 95 до 125 Вт, а межа споживання при короткочасних навантаженнях тепер може сягати (флагманська модель) до 250 Вт навіть з офіційної точки зору. Таке значне зростання апетитів не могло не позначитися на дизайні платформи в цілому: сумісні з Comet Lake-S моделі материнських плат з новим процесорним гніздом LGA1200, як правило, мають помітно потужніший, ніж раніше, конвертер живлення.

Але є й інша сторона: інженерам Intel довелося зробити спеціальні зусилля до того, щоб тепло, що виділяється кристалом Comet Lake-S, було відведено з належною ефективністю. У минулому поколінні процесорів, Coffee Lake-S, Intel

вдалася до збільшення теплопровідності застосовуваного термоінтерфейсного матеріалу під процесорною кришкою і перейшла з полімерної термопасти на індійний припій. Зараз зроблено ще один крок – з 800 до 500 мкм зменшено товщину процесорного кристала. Справа в тому, що вся напівпровідникова начинка розташована на кристалі з боку процесорної плати, а сам кремній має досить низьку теплопровідність, тому зменшення його шару дійсно сприяє покращенню тепловідведення.

Як видно з ілюстрації, витончення кристала супроводжується потовщенням мідної теплорозподільної кришки. Це зроблено для того, щоб зберегти товщину процесорів незмінною і щоб системи охолодження, розроблені для минулих поколінь CPU, залишилися сумісними з представниками сімейства Comet Lake-S.

Зростання споживаної новими процесорами потужності пов'язаний не тільки зі збільшенням розміру напівпровідникового кристала та з додатковими ядрами. Він викликаний ще й тим, що Intel додатково наростила тактові частоти і забезпечила нові чіпи хитрішими механізмами їх регулювання.

Наприклад, якщо порівняти старший 10-ядерник нового покоління зі старшим 8-ядерником минулого покоління, виявиться, що новинка виграє у попередника в ширині діапазону частот. Нині робочі частоти флагмана задаються інтервалом 3,7-5,3 ГГц, тоді як Core i9-9900K цей інтервал виглядав як 3,6-5,0 ГГц. Водночас реальні робочі частоти нових процесорів сильніше наблизилися до верхньої межі діапазону завдяки тому, що до стандартної та звичної технології Turbo Boost 2.0 додалися два нові «підсилювачі» – Turbo Boost Max 3.0 та Thermal Velocity Boost.

Технологія Turbo Boost 3.0 вже знайома нам процесорами для HEDT-систем: вона додає до стандартного турборежиму поняття «вдалих» ядер, які визначаються Intel індивідуально для кожного процесора на етапі виробництва. Сенс цього в тому, що при ізольованому навантаженні на ці ядра вони можуть прискорюватися на 100 МГц сильніше в порівнянні з іншими

ядрами. Передбачається, що диспетчер завдань операційної системи переноситиме на такі ядра малопотокові навантаження, дозволяючи отримувати додаткове поліпшення продуктивності порівняно з Turbo Boost 2.0. І цей механізм дійсно працює у версіях Windows 10, починаючи зі складання 1609. У сімействі процесорів Comet Lake-S «вдалих» ядер зазвичай два, але в цілому технологія Turbo Boost 3.0 доступна виключно для десяти-і восьмиядерних процесорів.

Інша реалізована в Comet Lake-S технологія – Thermal Velocity Boost – у процесорах для настільних систем зустрічається вперше і поки що доступна лише у старших моделях сімейства, що належать до серії Core i9. Вона відповідає за додаткове динамічне зростання частоти на 100 МГц - за межі заданих турборежимом кордонів. Як передбачається, такий «останній ривок» процесор може зробити за дотримання двох умов: якщо його споживання не виходить за певні специфікацією рамки і якщо температура не перевищує 70 градусів.

У сумі все це призводить до того, що старші десятиядерники можуть працювати на високих частотах, особливо при знятих межах споживання і при використанні якісних систем охолодження, здатних утримувати їх від перегріву.

Завдяки цим хитрощам нові процесори отримали більш високі частоти, ніж у восьмиядерних попередників. Причому більш високі частоти, порівняно з минулим флагманом Core i9-9900K, потенційно можуть забезпечити навіть 10-ядерні процесори нового покоління, не кажучи вже про восьмиядерників. Але є і винятки, наприклад Core i9-9900 виглядає за частотною формулою все-таки цікавіше, ніж новий Core i7-10700 з вісьмома ядрами, а випущений лімітованою серією Core i9-9900KS так і залишається єдиним CPU, здатним тримати частоту 5 ГГц при навантаженні відразу на всі ядра.

Нова платформа LGA1200 та набір системної логіки Z490

Як уже було сказано, процесори Comet Lake-S отримали нове виконання корпусу LGA1200 зі збільшенням контактів. І це означає, що вони не сумісні зі старими материнськими платами. Загалом це повністю відповідає стратегії Intel –

змінювати платформу кожні два покоління CPU, але в даному випадку перехід на оновлені материнські плати справді не позбавлений сенсу.

По-перше, між Comet Lake-S та їх попередниками існує помітна різниця в енергоспоживання, і для нових CPU дійсно потрібні потужніші схеми живлення, оскільки, навіть згідно з офіційними даними, у номінальному режимі вони можуть демонструвати споживання аж до 250 Вт, а в реальності ще вище, особливо під час розгону. Тому не дивно, що серед LGA1200-плат зустрічаються і такі, які оснащені 16-канальним конвертером живлення та силовими елементами, розрахованими на струм у 90 А. Також закономірно, що плати нового покоління хизуються більш просунутими системами охолодження VRM, які в деяких випадках можуть навіть забезпечуватись вентилятором.

По-друге, хоча про це поки не йдеться офіційно, у новій платформі Intel має намір запровадити підтримку PCI Express 4.0. Відбудеться це лише тоді, коли на ринку з'явиться наступне покоління процесорів Rocket Lake-S, але підготувати всю необхідну інфраструктуру Intel вирішила заздалегідь. Іншими словами, в платформі LGA1200 відразу закладено можливість переведення шини PCI Express у більш швидкі режими, хоча на даний момент у новій платформі всі слоти PCIe та M.2, як і раніше, максимально підтримуватимуть лише режим PCI Express 3.0.

Для платформи LGA1200 компанія Intel підготувала набори логіки чотириста серії, включаючи старший чіпсет Z490, який в контексті цієї статті нам найбільш цікавий. З точки зору характеристик материнські плати на базі цього чіпсету не дуже відрізняються від звичних плат на Z390, але головне - вони обладнуються новим процесорним роз'ємом LGA1200 і сумісні з Comet Lake-S, але не сумісні з Coffee Lake-S, як і з іншими процесорами попередніх поколінь.

У той же час важливо, що всі сьогоднішні LGA1200-плати гарантовано зможуть прийняти на борт і наступне покоління процесорів Rocket Lake-S, де знайде застосування нової мікроархітектури, PCI Express 4.0 та інтегрованої графіки Intel Xe. Вийдуть ці цікаві CPU, як очікується, наприкінці поточного чи,

скоріше, на початку наступного року. Хоча конкретні продажі можуть дещо відрізнятися, загальний підхід виробників плат полягає в тому, щоб використовувати в LGA1200-платах елементну базу (підсилювачі та комутатори), сумісну з четвертим поколінням PCI Express. Підтримка нового протоколу, мабуть, буде в майбутньому активована для першого слота PCIe x16, який логічно підключений до внутрішньопроцесорного контролера.

Якщо ж говорити про самий набір логіки Z490, то в його специфікаціях ви навряд чи зможете знайти якісь відмінності від Z390: чипсети ідентичні. Проте з погляду можливостей розширення материнські плати, побудовані навколо мікросхеми Z490, можуть мати характерні ознаки. Найперше з них слід визнати наявність 2,5-гігабітного мережного інтерфейсу – його можна виявити у більшості платформ верхньої та середньої цінових категорій. Також до списку стандартного обладнання для добротних LGA1200-плат входить бездротовий адаптер Wi-Fi 6 та збільшена кількість портів USB 3.2 Gen 2 (10 Гбіт/с).

Варто згадати, що разом з Z490 на ринок приходять і два простіші набори логіки: B460 і H410. Вони не мають функцій розгону процесора, бідніших по оснащенню, а плати на їх основі вже не зможуть похвалитися такими ж потужними стабілізаторами живлення. Проте докладніше про них ми поговоримо пізніше, коли займатимемося тестами недорогих представників сімейства Comet Lake-S.

Подробиці про Core i9-10900K

Для тестування нам удалося отримати старший процесор у сімействі Comet Lake-S, десятиядерник Core i9-10900K. Відразу потрібно сказати, що отриманий нами зразок – це серійний екземпляр актуального степпіга Q0, і саме його очікування затримало цей огляд. За деякий час до анонсу Intel пропонувала для тестів процесори більш раннього степінгу P1, але він зрештою серійні вироби не пішов. Тому ми вирішили не тестувати попередній зразок і дочекатися правильного, тим більше, що, як потім з'ясувалося, процесори попереднього степінгу дійсно відрізняються від фінальних: вони використовують завищену

напругу, сильніше гріються і не досягають максимальних частот у турборежимі.

Характеристики Core i9-10900K можна переглянути на скріншоті CPU-Z, наведеному нижче. Головне, що потрібно знати про цей процесор: у ньому є 10 ядер за допомогою технології Hyper-Threading, 20 Мбайт L3-кешу, а робочі частоти знаходяться на околиці 5,0 ГГц. При цьому тепловий пакет збільшений до 125 Вт при дозволеному піковому споживанні до 250 Вт. Тобто це швидкий, але й гарячий процесор.

Формально максимальні частоти, дозволені турборежимом залежно від навантаження на різну кількість обчислювальних ядер, перевищують 4,8 ГГц:

Але треба розуміти, що у реальному житті частоти, швидше за все, обмежуватимуться не цими показниками, а межами енергоспоживання: 125 Вт для довготривалих навантажень (PL1) та 250 Вт – для короткочасних (PL2). Майте на увазі, що оголошена в специфікації величина TDP 125 Вт – це, згідно з визначенням Intel, середнє значення тепловиділення при складному навантаженні на всі ядра, коли процесор працює на базовій частоті 3,7 ГГц. Тому насправді частоти сильно не дотягуватимуть до тих, що зазначені вище. Крім того, табличні значення вказані з урахуванням Thermal Velocity Boost, тобто припущення, що температура процесора не перевищує 70 градусів. В іншому випадку вони будуть на 100 МГц нижче.

Якщо не порушувати специфікацій Intel з енергоспоживання, то при тривалих багатопотокових навантаженнях Core i9-10900K повинен працювати на частотах близько 4,0 ГГц. А заявлені 4,9 ГГц він може взяти лише за короткочасних багатопоточних навантажень, коли для енергоспоживання діє подвоєна щодо паспортного TDP межа. З іншого боку, максимальне значення частоти для однопоточного навантаження – 5,3 ГГц – практично досягається не часто. Воно можливе лише при складанні зусиль технологій Turbo Boost Max 3.0 та Thermal Velocity Boost і тому спостерігається епізодично. Середня частота Core i9-10900K при навантаженні на одне-два ядра, що спостерігається на практиці, ближче до 5,2 ГГц. Але навіть незважаючи на деякий розрив між

теоретичними і показниками, що спостерігаються, частоти Core i9-10900K не розчаровують:

На додаток до ядра, що додалися, і підвищених частот в процесорах Comet Lake-S з'явилася офіційна підтримка DDR4-2933 SDRAM. Це має значення для користувачів, які захочуть скористатися материнськими платами на базі молодших чіпсетів – тепер там пам'ять зможе працювати на більш високих швидкостях. Що ж до Z490, то платах на його основі розгін пам'яті необмежений і можливий до дуже високих значень.

Рекомендована ціна Core i9-10900K складає $488, версія цього ж процесора із заблокованим GPU (Core i9-10900KF) обійдеться у $472. Це означає, що Intel вирішила протиставити свій новий десятиядерник дванадцятиядерний процесор AMD Ryzen 9 3900X з офіційною ціною $499. Якщо зіставити характеристики цих двох CPU, то картина виходить наступною.

Ryzen 9 3900X в порівнянні з Core i9-10900K пропонує більш розвинену багатопоточність, у нього на 20% більше обчислювальних ядер, і це відображає парадигму AMD, яка наголошує на постійне збільшення можливостей паралельної обробки. Підхід Intel інший: "синя" компанія все ще продовжує приділяти істотну частину своєї уваги піковим однопоточних швидкостей. І хоча Intel певною мірою слідує прикладу AMD і її флагман в черговий раз отримав додаткові ядра, головна його перевага - у максимальній тактовій частоті, яка вище, ніж у процесора AMD, що конкурує, на 15%. Щоправда, не варто забувати, що на користь AMD тут може зіграти деяку перевагу в IPC: мікроархітектура Zen 2 за цим показником трохи оминула Skylake. Але сумарно, дивлячись на характеристики, цілком можна стверджувати, що процесори Ryzen 9 3900X і Core i9-10900K заслуговують на те, щоб виступати в одній ваговій категорії. Хоча не можна заперечувати, що Ryzen 9 3900X має невелику формальну перевагу завдяки підтримці PCI Express 4.0 і кращій економічності.

Температура та споживання

Споживання та нагрівання для Core i9-10900K – це, очевидно, болюче

питання. Досі Intel не мав таких багатоядерних і високочастотних процесорів, але вже за минулим поколінням масових CPU було зрозуміло, що мікроархітектура Skylake при наближенні до 5 ГГц втрачає всякі натяки на енергоефективність. Олії у вогонь підливають численні публікації, в яких йдеться про реальне споживання нового десятиядерника на рівні 250, а то й 300 Вт. І на цьому етапі у багатьох виникає непорозуміння, чому Intel говорить про показник TDP 125 Вт, і чи тут немає явного обману.

Але насправді все чесно. Справа в тому, що існує два базові варіанти налаштування будь-якого процесора Intel: з прицілом на дотримання заявлених параметрів енергоспоживання та тепловиділення та з прицілом на максимальну продуктивність. У випадку з Core i9-10900K перший варіант передбачає, що шляхом динамічного підстроювання частоти фактичне споживання процесора утримується в рамках величини PL1, встановленої для нього в 125 Вт, і виходити за цей кордон і підвищувати споживання до PL2, що дорівнює 250 Вт, йому дозволяється лише короткочасно. Другий варіант передбачає, що ніякі межі споживання PL1 і PL2 не діють, і процесор завжди функціонує на максимальній частоті, визначеної турборежимом для даної кількості завантажених роботою ядер. Перемикання між цими варіантами виконується в BIOS материнських плат.

У цих двох підходів немає жодної новини – так було й раніше. Просто з процесорами Intel минулих поколінь виробники материнських плат особливо не замислювалися про дотримання якихось там обмежень і за умовчанням включали режим максимальних частот (за цією функцією закріпилася назва MCE Multi-Core Enhancements). Але тепер так уже не вийде: тепловиділення Core i9-10900K зі знятими обмеженнями може виявитися настільки високим, що процесор перегріватиметься навіть із досить потужними серійними системами охолодження. До того ж Intel забрала у виробників материнських плат можливість відсувати кордон максимально дозволеного нагрівання кристала, і тепер вона знову жорстко встановлена в 100 градусів, тоді як для процесорів Coffee-Lake-S її майже завжди примусово відсували до 115 градусів.

В результаті плати для процесорів Comet Lake-S, швидше за все, пропонуватимуть якісь компроміси щодо динамічного управління частотами. Принаймні, ми сподіваємося, що здоровий глузд візьме гору і виробники не будуть без попередження відправляти частоти нових процесорів в стратосферу, прирікаючи користувачів на зустріч з перегрівом, тротлінгом і тому подібними речами, які легко можуть зіпсувати враження про платформу LGA1200. Наприклад, тестуючи Core i9-10900K на платі ASUS Maximus XII Hero (Wi-Fi), ми побачили таку реалізацію: при першому старті системи після встановлення нового процесора BIOS пропонує вибір: або використовувати надалі максимальні частоти, або все-таки слідувати специфікаціям Intel.

І це – розумний підхід: користувач повинен розуміти, що скасування меж споживання PL1 і PL2 схоже на розгін, і стабільно працювати в цьому випадку процесор все-таки не зобов'язаний.

Але давайте проаналізуємо, наскільки ці два режими налаштування процесора (номінальний та безлімітний) розрізняються за реальними частотами, продуктивністю, температурами та споживанням. Ми провели їхнє практичне порівняння на прикладі навантаження Cinebench R20 та Prime 95 AVX2 29.8 з використанням для охолодження процесора суперкулера Noctua NH-D15.

Реальне довготривале енергоспоживання десятиядерного процесора при багатопоточному навантаженні, коли він працює на частотах, заявлених у специфікації, але без будь-яких обмежень, може становити і 230 Вт (Cinebench R20), і навіть 300 Вт (в Prime95). Звичайно, включення в частотну формулу відсічення частоти при зростанні споживання вище 125 Вт змінює все кардинально.

Можна сказати, що «чесний» Core i9-10900K і «безлімітний» Core i9-10900K – це два різні процесори, різниця між якими буде проявлятися тим сильніше, чим складніше і триваліше виконувані завдання. У той же час на відносно коротких навантаженнях, протягом яких для процесора діє підвищений енергоспоживання PL2, режим з активованими обмеженнями майже не знижує

продуктивність процесора. Максимальний час дії цієї межі замість консервативного PL1 також визначено специфікацією і Core i9-10900K становить 56 секунд. Однак на практиці проміжки високого споживання не перевищують 30-40 секунд, і то за умови, що раніше процесор перебував у стані простою. Пояснюється це тим, що, крім контролю миттєвого рівня споживання,

Перспектива використовувати Core i9-10900K без урахування всіх цих численних обмежень виглядає привабливо. Однак проблема полягає в тому, що «безлімітний» режим змушує користувача зіткнутися з високими температурами, що лякають.

Навіть із потужним двобаштовим кулером у Cinebench R20 процесор у «безлімітному» режимі може нагріватися до 83 градусів, а в важчому в обчислювальному плані тесті Prime95 температура може сягати 93 градусів, що досить близько до граничного значення. Грунтуючись на цих даних, логічно припустити, що серед Core i9-10900K можуть бути такі екземпляри, які не зможуть функціонувати без тротлінгу в режимі зі знятими обмеженнями.

Однак тут потрібно вставити важливу ремарку про те, що якщо Core i9-10900K буде використовуватися в системі, спрямованій виключно на ігрове застосування, і він ніколи не стикатиметься з додатками для обробки контенту, то безлімітний режим цілком допустимий. Сучасні ігри не створюють таких навантажень, які були б здатні повністю задіяти всі ресурси десятиядерного CPU під важкі обчислювальні алгоритми, тому нагрівання та споживання у них помітно нижче.

На прикладі Shadow Of The Tomb Raider ми бачимо вдвічі нижче енергоспоживання порівняно з рендерингом і, відповідно, цілком комфортні для процесора температури на рівні 50-60 градусів як у номінальному, так і безлімітному режимі.

Розгін

На цей момент вже зрозуміло, що на якийсь значний розгін Core i9-10900K можна не розраховувати. Чого можна очікувати від процесора, який

навіть на штатних частотах (при скасуванні меж споживання) нагрівається до 92-93 градусів за допустимого максимуму в 100?

У тому, що Core i9-10900K дуже важко зрушити з номінальних частот нагору, ми легко переконалися на практиці. Максимальна частота, з якою нам вдалося пройти між Сциллою перегріву та Харибдою нестабільності, склала 5,1 ГГц за умови додавання коригуючого коефіцієнта -2 для процесорного множника при AVX-навантаженні. Іншими словами, процесор зміг працювати при 5,1 ГГц за звичайного обчислювального навантаження і проходив тести стабільності, демонструючи температури не вище 95 градусів.

Але при AVX-навантаженні частоту доводилося скидати до 4,9 ГГц, щоб уникнути перегріву. Однак навіть на такій частоті його температура сягала 96 градусів. Для досягнення описаного результату в BIOS материнської плати ASUS Maximus XII Hero (Wi-Fi) напруга процесора була знижена на 0,035 В щодо номіналу (через налаштування Offset) з одночасною активацією функції Load-Line Calibration в режимі Level 5. Внаслідок застосування цих налаштувань реальна напруга CPU при стресовому навантаженні встановлювалося лише на рівні 1,16-1,24 В.

Відведення тепла в нашому випадку забезпечувало суперкулер Noctua NH-D15, до ефективності якого важко пред'явити якісь претензії. А це означає, що радикальним оверклокерам, які звикли не задовольнятися номінальним або близьким до номінального режиму роботи процесора, доведеться націлюватися на оснащення своїх збірок якимось більш потужним охолодженням, наприклад самостійно зібраними системами рідинного охолодження.

У цьому випадку їм також можуть стати в нагоді нові тонкі оверклокерські інструменти, які з'явилися в Core i9-10900K і в платформі LGA1200 загалом. Зокрема, для цього процесора можна не лише задавати свій власний множник для різної кількості навантажених роботою ядер, а й поядерно відключати технологію Hyper-Threading.

Друге цікаве нововведення – повний доступ до кривої залежності між

напругою процесора та його частотою.

Все це зрештою відкриває величезний простір для оптимізації налаштувань системи, але, на жаль, не скасовує того факту, що Core i9-10900K працює майже на межі можливостей 14-нм кремнію, і навіть для незначного розгону спочатку доведеться вирішити задачу з відведенням від чіпа понад 300 Вт тепла.

Дамо опис алгоритму AES. У його основі лежить алгоритм Rijndael

Формат блоків даних і число раундів

RIJNDAEL – це симетричний блоковий шифр, що оперує блоками даних розміром 128 і довжиною ключа 128, 192 або 256 біт – назва стандарту відповідно "AES-128", "AES-192", і "AES-256".

Уведемо наступні позначення:

– Nb – число 32-бітних слів які містяться у вхідному блоці, Nb – 4;

– Nk – число 32-бітних слів, що втримуються в ключі шифрування, Nk = 4,6,8;

– Nr – число раундів шифрування, як функція від Nb і Nk, Nr = 10,12,14.

Вхідні (input), проміжні (state) і вихідні (output) результати перетворень, виконуваних у рамках криптоалгоритму, називаються станами (State). Стан можна представити у вигляді матриці 4 x Nb, елементами якої є байти (чотири рядки по Nb байт) у порядку S00, S10, S20, S30, Sm, Sn, S21, S31. Рисунок 3.1 демонструє таке подання, що носить назву архітектури «Квадрат».

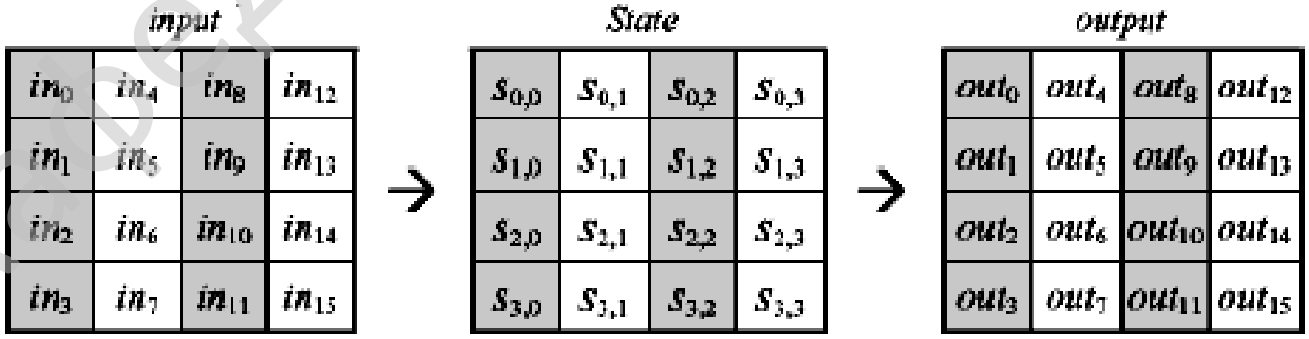

Рисунок 3.1 – Приклад подання блоку у вигляді матриці 4xNb

На старті процесів шифрування й дешифрування масив вхідних даних те, input перетвориться в масив State за правилом:

$$s[r,c] = in[r + 4c], \quad (3.1)$$

де $0 < r < 4$ і $0 < c < Nb$.

Наприкінці дії алгоритму виконується зворотне перетворення:

$$out[r + 4c] = s[r,c], \quad (3.2)$$

де $0 < r < 4$ і $0 < c < Nb$ – вихідні дані виходять із байтів стану в тому же порядку.

Чотири байти в кожному стовпці стану являють собою 32-бітне слово, якщо r – номер рядка в стані, те одночасно він є індексом кожного байта в цьому слові. Отже стан можна представити як одномірний масив 32-бітних слів $w_0,...,w_n$, де номер стовпця стану с є індекс у цьому масиві. Тоді стан можна представити так:

$w_0 = s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}$	$w_2 = s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}$
$w_1 = s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}$	$w_3 = s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}$

Ключ шифрування також як і масив State представляється у вигляді прямокутного масиву із чотирма рядками. Число стовпців цього масиву дорівнює Nk.

Для алгоритму AES число раундів Nr визначається на старті залежно від значення Nk (таблиця 3.1):

Таблиця 3.1 – Залежність значення Nr від Nk і Nb

	Nk	Nb	Nr
AES– 128	4	4	10
AES– 192	6	4	12
AES – 256	8	4	14

Алгоритм вироблення ключів (Key Schedule)

Введемо наступні позначення:

– Rcon[i] – масив 32-бітних раундових констант;

– RotWord() – операція циклічної перестановки вхідного 4-х байтного слова у вихідне за наступним правилом [a0, ai, a2, a3] —> [ax, a2, a3, a0];

– SubWord() – операція заміни в 4-х байтному слові за допомогою S-Box кожного байта;

⊕ – операція що виключає або – XOR.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

Раундові ключі виходять із ключа шифрування за допомогою алгоритму вироблення ключів. Він містить два компоненти: розширення ключа (Key Expansion) і вибір раундового ключа (Round Key Selection). Основні принципи алгоритму виглядають у такий спосіб:

– загальне число бітів раундових ключів дорівнює довжині блоку, помноженої на число раундів, плюс 1;

– ключ шифрування розширюється в розширений ключ (Expanded Key);

– раундові ключі беруться з розширеного ключа в такий спосіб: перший раундовий ключ містить перші Nb слів, другий – наступні Nb слів і т.д.

Розширення (планування) ключа

Розширений ключ являє собою лінійний масив w[i] складається з Nb*(Nr + 1) 4-х байтових слів, i = Nb*(Nr + 1).

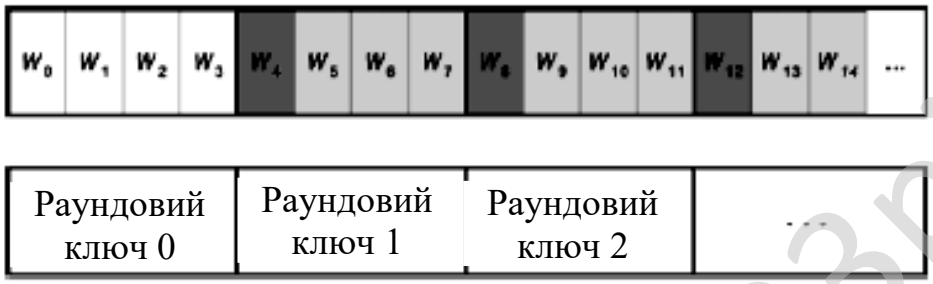

Рисунок 3.2 – Процедури розширення й виборки раундового ключа для Nk = 4

Ясно-сірим кольором виділені слова розширеного ключа, які формуються без використання функцій SubWord() і RotWord().

Темно-сірим кольором, виділені слова розширеного ключа, при обчисленні яких використовуються перетворення SubWord() і RotWord().

Перші Nk слів містять ключ шифрування. Кожне наступне слово w[i] виходить за допомогою XOR попереднього слова w[i-1] і слова на Nk позицій раніше:

$$W[i\text{-}Nk]: w[i]= w[i\text{-}1] \oplus w[i\text{-}Nk]. \quad (3.3)$$

Для слів, позиція яких кратна Nk, перед XOR застосовується перетворення до w[i-1], а потім ще додається раундова константа Rcon[i]. Перетворення реалізується за допомогою двох додаткових функцій: RotWord() і SubWord().

Значення Rcon[j] дорівнює 2j-1. Значення w[i] у цьому випадку визначається вираженням:

$$w[i] = SubWord(RotWord(w[i\text{-}1]) \oplus Rcon[i/Nk] \oplus M[i\text{-}Nk]. \quad (3.4)$$

Вибір раундового ключа і-тий раундовий ключ вибирається зі слів масиву розширеного ключа в проміжку від W[Nb * i] до W[Nb * (i+1)].

Для функції зашифрування розширений ключ генерується алгоритмом. Для функції зворотного розшифрування використовується цей же ключ, але у зворотній послідовності починаючи з останнього раундового підключа зашифрування.

Для функції прямого розшифрування використовується трохи модифікований алгоритм планування ключа. При формуванні розгорнутого ключа в процедуру планування необхідно додати наприкінці додаткове перетворення, причому розширений ключ використовується при прямому розшифруванні в тій же послідовності, що й при зашифруванні.

Додаткове перетворення розширеного ключа для функції прямого розшифрування:

```
for i = 0 step 1 to (Nr+1)*Nb-1
   dw[i] = w[i]
end for

for round = 1 step 1 to Nr-1
   InvMixColumns(dw[round*Nb, (round+1)*Nb-1])
end for
```

Функція зашифрування

Введемо наступні позначення:

– SubBytes() – заміна байтів – побайтова нелінійна підстановка в State-блоках (S-Box) з використанням фіксованої таблиці замін розмірністю 8x256 (affain map table).

– ShiftRows() – зрушення рядків – циклічне зрушення рядків масиву State на різну кількість байт.

– MixColumns() – перемішування стовпців – множення стовпців стану, розглянутих як багаточлени над GF(2^8).

– AddRoundKey() – додавання з раундовим ключем – поразрядне XOR умісту State з поточним фрагментом розгорнутого ключа.

На псевдокоді операція зашифрування виглядає в такий спосіб:

```
Cipher[byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)]]
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])

   for round = 1 step 1 to Nr-1
      SubBytes(state)
      ShiftRows[state)
      MixColumns(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Після заповнення масиву State елементами вхідних даних до нього застосовується перетворення AddRoundKey, далі, залежно від величини Nk, масив State піддається раундовій трансформації 10, 12 або 14 разів, причому у фінальний раунд є трохи вкороченим – у ньому відсутнє перетворення MixColumns. Вихідними даними описаної послідовності операцій є шифротекст – результат дії функції зашифрування AES.

Функції розшифрування

У специфікації алгоритму AES пропонуються два види реалізацій функції розшифрування відмінних друг від друга послідовністю додатка перетворень зворотних перетворенням функції зашифрування й послідовністю планування ключів.

Введемо наступні позначення:

– InvSubBytes() – зворотна SubBytes() заміна байтів – побайтова нелінійна підстановка в S-блоках з використанням фіксованої таблиці замін розмірністю 8x256 (inverse affain map).

– InvShiftRows() – зворотне зрушення рядків ShiftRows() – циклічне зрушення рядків масиву State на різну кількість байт.

– InvMixColumns() – відновлення значень стовпців – множення стовпців стану, розглянутих як багаточлени над GF(2^8).

Функція прямого розшифрування

Змінивши певним чином послідовність планування ключа можна побудувати ще один алгоритм – алгоритм прямого розшифрування.

```
EqInvCipher(byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvSubBytes(state)
      InvShiftRows(state)
      InvMixColumns(state)
      AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
   end for

   InvSubBytes(state)
   InvShiftRows(state)
   AddRoundKey(state, dw[0, Nb-1])

   out = state
end
```

Дві наступні властивості дозволяють зробити це:

– Порядок додатка функцій SubBytes() і ShiftRows() не грає ролі. Те ж саме вірно й для операцій InvSubBytes() і InvShiftRows(). Це відбувається тому, що функції SubBytes() і InvSubBytes() працюють із байтами, а операції ShiftRows() і InvShiftRows() зрушують цілі байти, не зачіпаючи їхніх значень.

– Операція MixColumns() є лінійною щодо вхідних даних, що означає InvMixColumns(State XOR RoundKey) = InvMixColumns(State) XOR InvMixColumns(RoundKey).

Ці властивості функцій алгоритму шифрування дозволяють змінити порядок застосування функцій InvSubBytes() і InvShiftRows(). Функції AddRounKey() і InvMixColumns() також можуть бути застосовані у зворотному порядку, але за умови, що стовпці (32-бітні слова) розгорнутого ключа розшифрування попередньо пропущені через функцію InvMixColumns().

Таким чином, можна реалізувати більше ефективний спосіб розшифрування з тим же порядком додатка функцій як і в алгоритмі зашифрування.

Функція зворотного розшифрування

Якщо замість SubBytes(), ShiftRows(), MixColumns() і AddRoundKey() у зворотній послідовності виконати інверсні їм перетворення, можна побудувати функцію зворотного розшифрування. При цьому порядок використання раундових ключів є зворотним стосовно тому, що використовується при зашифруванні. На псевдокоді вона виглядає так:

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)
      InvSubBytes(state)
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Алгоритм зворотного розшифрування, описаний вище має порядок додатка операцій – функцій зворотний порядку операцій в алгоритмі прямого зашифрування, але використовує ті ж параметри розгорнутого ключа.

Раундове перетворення

Раундове перетворення складається з послідовного застосування до масиву State ряду трансформацій.. Зараз обговоримо деталі його реалізації.

Нелінійна заміна байтів масиву стану за допомогою трансформації SubBytes має вигляд:

$$\lambda(f) = ((X^4 + X^3 + X^2 + X + 1) \cdot f + X^6 + X^5 + X + 1) \bmod (X^8 + 1) \quad (3.5)$$

Багаторазове обчислення в процесі зашифрування даного вираження робило б невиправдане обчислювальне навантаження на виконуючу систему, тому для практичної реалізації найбільш прийнятним рішенням є використання

попереднє обчисленої таблиці заміни S-Box. Логіка роботи S-Box при перетворенні байта (xy} відбита в шестнадцятковому виді на рисунку 3.3:

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
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

Рисунок 3.3 – Таблиця S-Box заміни байт

Її використання зводить операцію SubBytes до найпростішої вибірки байта з масиву λ (f) = Sbox[f].

У функціях розшифрування застосовується операція зворотна InvSubBytes().

Вона реалізується так само просто, як і попередня за допомогою інверсної таблиці S-Box – λ^{-1}(f) = InvSbox[f]. Її логіка роботи при перетворенні байта (xy} відбита в шестнадцятковому виді на рисунку 3.4.

Рисунок 3.5 ілюструє застосування перетворення заміни байт до стану у функціях зашифрування й розшифрування.

У перетворенні зрушення рядків (ShiftRows) останні 3 рядка стану циклічно зрушуються ВЛІВО на різне число байтів. Рядок 1 зрушується на C1 байт, рядок 2 – на C2 байт, і рядок 3 – на C3 байт. Значення зрушень C1, C2 і C3 в Rijndael залежать від довжини блоку Nb .

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

Рисунок 3.4 –Таблиця S-Box інверсної заміни байт

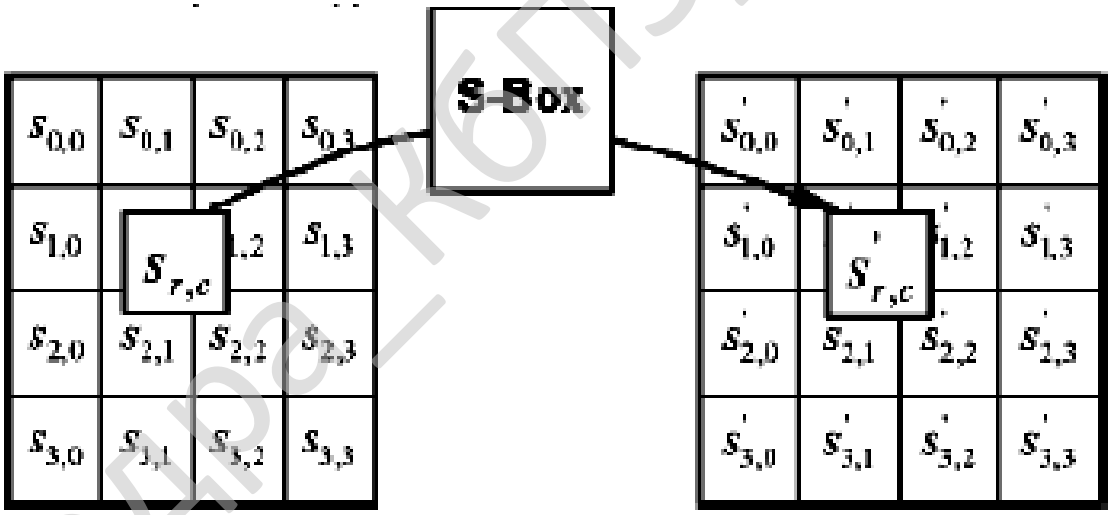

Рисунок 3.5 – Перетворення стану за допомогою таблиці заміни S-Box

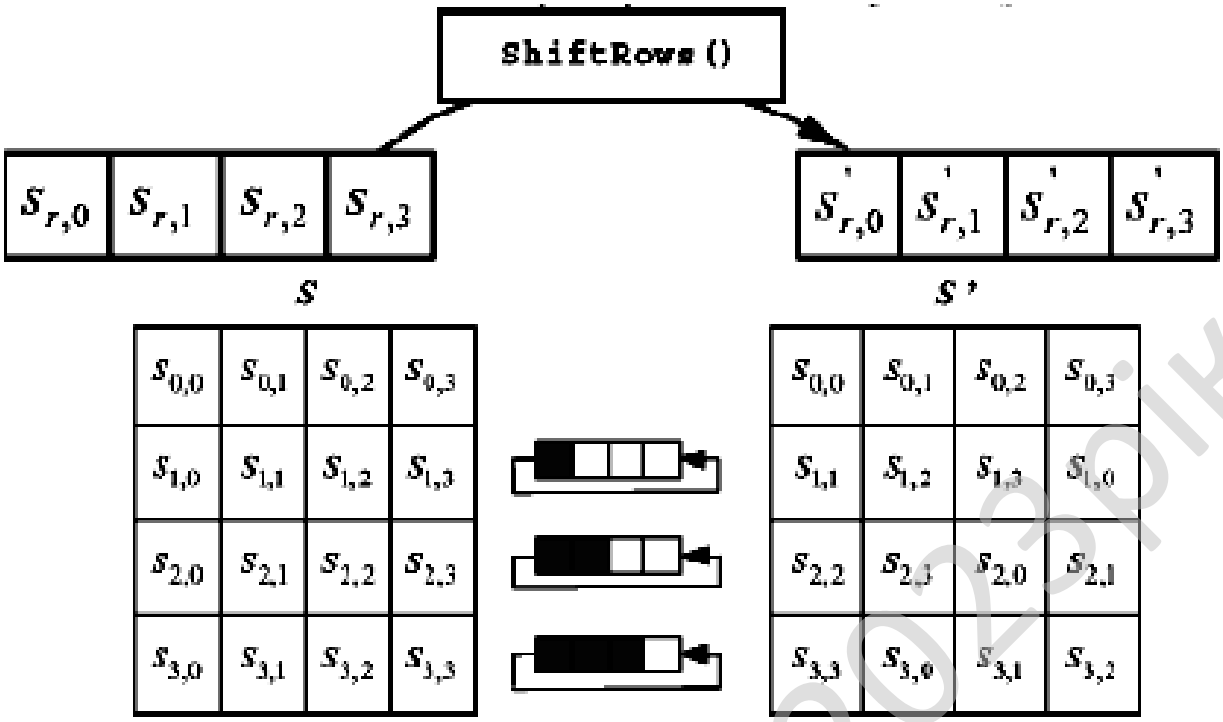

Рисунок 3.6 – Перетворення зрушення рядків у функції зашифрування

У перетворенні зворотного зрушення рядків InvShiftRows останні 3 рядка стану циклічно зрушуються ВПРАВО на різне число байтів. Рядок 1 зрушується на C1 байт, рядок 2 – на C2 байт, і рядок 3 – на C3 байт.

Перемішування стовпців

У перетворенні перемішування стовпців (MixColumns) стовпці стану розглядаються як багаточлени над GF(28) і піддаються перетворенню c(x) = c * gmod(Y^4 +1), де c = (X,1,1,X+1), тобто множаться за модулем $x^4 + 1$ на багаточлен c(x), що виглядає, як: $c(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$.

Застосування цієї операції до всім чотирьох стовпцям стану позначається, як MixColumns(State). Рисунок 3.7 демонструє застосування перетворення MixColumns до стовпця стану.

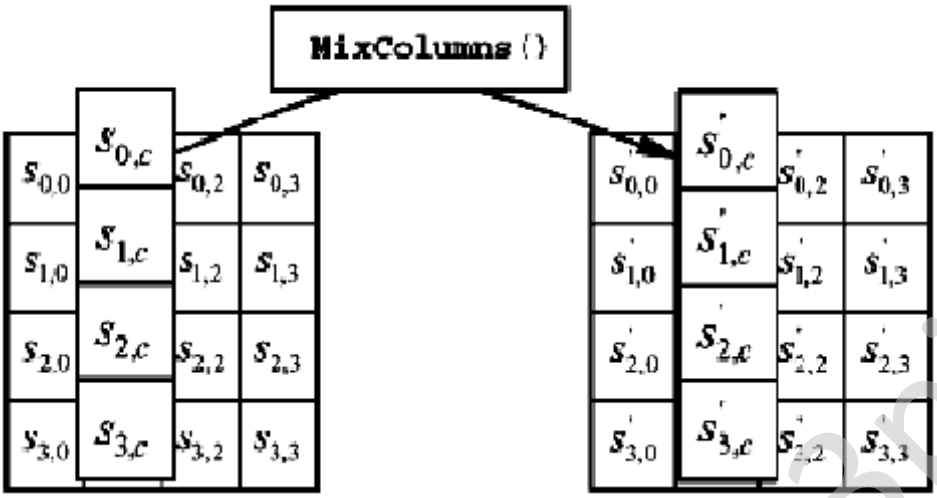

Рисунок 3.7 – Операція перемішування діє на стовпці стану

У зворотному перетворенні InvMixColumns стовпці стану розглядаються як багаточлени над GF(2^8), але, природно, піддаються зворотному перетворенню.

Додавання раундового ключа AddRoundKey()

У даній операції раундовий ключ додається до стану за допомогою поразрядного XOR. Довжина ключа (в 32-розрядних словах) дорівнює довжині блоку Nb . Рисунок 3.8 ілюструє дію даної операції на стан. Це перетворення позначається як AddRoundKey(State, RoundKey).

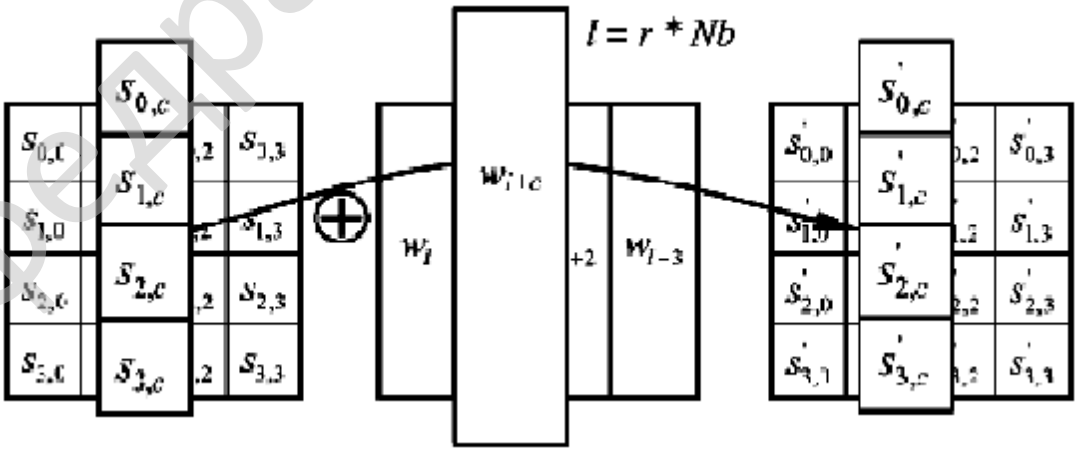

Рисунок 3.8 – AddRoundKey

Основні особливості AES

У висновку сформулюємо основні особливості AES:

– нова архітектура «Квадрат», що забезпечує швидке розсіювання й перемішування інформації, при цьому за один раунд перетворенню піддається весь вхідний блок;

– байт-орієнтована структура, зручна для реалізація на 8– розрядні мікро контролерах;

– всі раундові перетворення суть операції в кінцевих полях, що допускають ефективну апаратну й програмну реалізацію на різних платформах.

3.2 Розробка структурної схеми

На рисунку 3.9 зображена структурна схема системи зберігання конфіденційних даних з розмежованим доступом.

Виходячи зі структурної схеми системи зображеної на рисунку 3.9, система зберігання конфіденційних даних з розмежованим доступом, працює наступним чином.

Спершу при вході в систему, користувач звертається до блоку розмежування доступу.

Блок розмежування доступу отримує пароль користувача, та звертається до менеджеру паролів, де отримує сеансовий пароль перевірки правильності паролю користувача, та правильності прав доступу користувача, які зберігаються у відповідних зашифрованих базах даних.

Розмежування цих баз зроблено з метою підвищення стійкості системи зберігання інформації.

Після підтвердження прав доступу, та правильності введеного паролю, користувачеві видається сеансовий ключ AES з використанням спеціальних інструкцій Intel Core i9-10900K для роботи з інформацією.

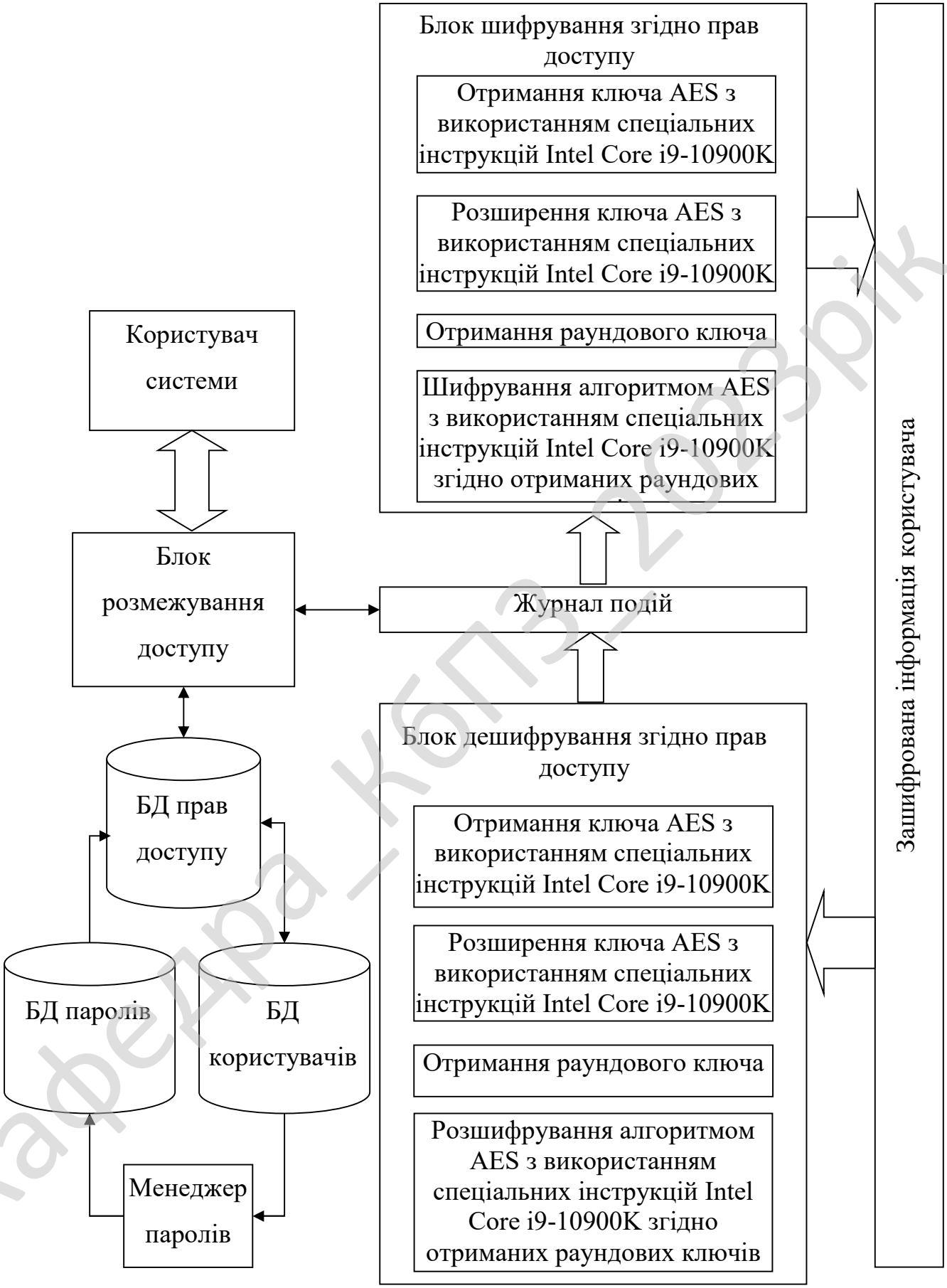

Рисунок 3.9 – Структурна схема системи

У блоці шифрування згідно прав доступу, з отриманого ключа AES з використанням спеціальних інструкцій Intel Core i9-10900K відбувається його розширення, та обирається ключ ітерації, за допомогою яких й відбувається шифрування інформації алгоритмом AES з використанням спеціальних інструкцій Intel Core i9-10900K.

Процедура дешифрування відбувається аналогічним чином.

3.3 Розробка функціональної схеми

На рисунку 3.10 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

Функціональна схема складається з наступних блоків:

– Головне вікно програми.

– Блоки шифрування та дешифрування інформації згідно алгоритму AES з використанням спеціальних інструкцій Intel Core i9-10900K.

– Блок розмежування доступу.

– Блок менеджеру паролів.

– Блок журналювання подій.

– Допомога.

Розглянемо ці блоки більш детально.

Головне вікно програми

Головне вікно призначене для швидкого доступу до основних функцій програми й меню. Програма складається з головного вікна, розташованого у верхній частині екрана й набору незалежних дочірніх вікон. Розташування й розміри вікон можна змінювати за допомогою миші. Також існує можливість закрити непотрібні дочірні вікна (знову відобразити їх можна шляхом вибору відповідних пунктів у меню натисканням на аналогічні кнопки в головному вікні програми). Всі зроблені зміни збережуться в наступному сеансі роботи. Призначення всіх кнопок у програмі пояснюється спливаючими підказками: підведіть покажчик миші до будь-якої кнопки й затримаєте його – з'явиться спливаюча підказка із призначенням кнопки. Головне меню надає доступ до основних списків і функцій системи.

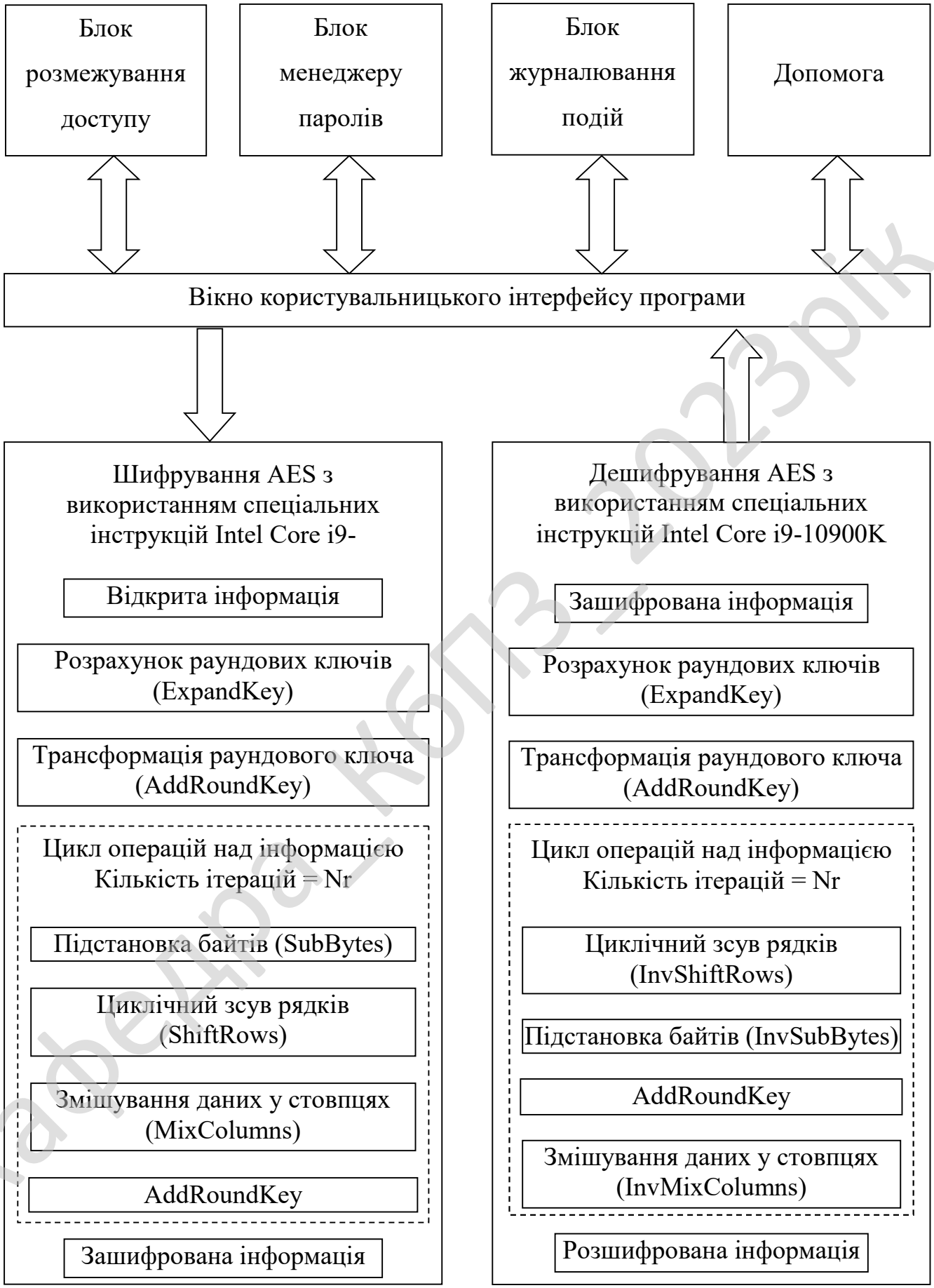

Рисунок 3.10 – Функціональна схема системи

Блок розмежування доступу

Призначений для організації безпечного доступу співтовариства користувачів до захищених ресурсів. Члени цього співтовариства, використовуючи програму, одержують визначені переваги. Це дає наступні можливості:

– Надавати користувачам доступ до інформації (наприклад, групи структурних схем, адресні довідники відділів або пошук співробітників) і ресурсам (наприклад, устаткування або облікові записи у внутрішніх системах), у яких вони бідують, буквально з першого дня.

– Синхронізувати кілька паролів з одним ім'ям користувача для всіх систем.

– При необхідності оперативно змінювати або відзивати права на доступ (наприклад, при переході співробітника в іншу групу або при звільненні).

– Підтримувати відповідність урядовим постановам.

У цей блок включені наступні можливості.

Ролі, що дозволяють виконувати наступні дії:

– запитувати призначення ролей і управляти процесом підтвердження запитів на призначення ролей;

– перевіряти стан Ваших запитів ролей;

– визначати ролі і їхні взаємини;

– визначати обмеження поділу обов'язків (SoD) і управляти процесом підтвердження у випадках, коли користувач запитує перевизначення обмеження;

– переглядати довідник ролей;

– переглядати докладні звіти, у яких перераховані ролі й обмеження поділу обов'язків, визначені в довіднику, а також поточний стан призначення ролей, виключення поділу обов'язків і повноваження користувача.

Модуль "Дотримання" дозволяє:

– Запитувати підтвердження профілю користувача.

– Запитувати підтвердження поділу обов'язків (SoD).

– Запитувати підтвердження призначення функцій.

– Запитувати підтвердження призначення користувача.

Самообслуговування облікового запису, що дозволяє:

– відображати структурні схеми;

– повідомляти про додатки, пов'язані з користувачем, для адміністратора;

– змінювати дані профілю;

– виконувати пошук у каталозі;

– змінювати пароль, відповідь на запит-відповідь пароля і його підказку;

– переглядати стан політики й синхронізації пароля;

– створювати облікові записи для нових користувачів і груп (при наявності відповідних повноважень).

Запити й твердження, що дозволяють:

– запитувати ресурси;

– перевіряти підтвердження запитів на ресурси;

– працювати із призначеними завданнями підтвердження інших запитів на ресурси;

– виконувати запити й твердження в якості чиєїсь довіреної особи або делегата;

– призначати кого-небудь ще довіреною особою або делегатом (при наявності відповідних повноважень);

– управляти всіма цими функціями запитів і підтверджень в інтересах Вашої групи (при наявності відповідних повноважень);

– при необхідності для кожного запиту або підтвердження надавати цифровий підпис.

Блоки шифрування та дешифрування інформації згідно алгоритму AES з використанням спеціальних інструкцій Intel Core i9-10900K

Призначені для шифрування та дешифрування інформації, до якої користувач має доступ, згідно прав доступу.

При реалізації шифрування та дешифрування виконуються такі основні операції:

– Key Expansion – процедура використовувана для генерації Round Keys з Cipher Key.

– Cipher Key – секретний, криптографічний ключ, що використовується Key Expansion процедурою, щоб зробити набір ключів для раундів (Round Keys); може бути представлений як прямокутний масив байтів, що має чотири рядки й Nk колонок.

– Round Key – Round Keys виходять із Cipher Key використовуючи процедуру Key Expansion. Вони застосовуються до State при шифруванні й розшифруванні.

– State – проміжний результат шифрування, що може бути представлений як прямокутний масив байтів що має 4 рядки й Nb колонок.

– AddRoundKey() – трансформація при шифруванні й зворотному шифруванні, при якій Round Key XOR'иться с State. Довжина RoundKey дорівнює розміру State (тобто, якщо Nb = 4, то довжина RoundKey дорівнює 128 біт або 16 байт).

– SubBytes() – трансформації при шифруванні які обробляють State використовуючи нелінійну таблицю заміщення байтів (S-box), застосовуючи її незалежно до кожного байта State.

– ShiftRows() – трансформації при шифруванні, які обробляють State, циклічно зміщаючи останні три рядки State на різні величини.

– MixColumns() – трансформація при шифруванні яка бере всі стовпці State і змішує їх дані (незалежно друг від друга), щоб одержати нові стовпці.

– InvShiftRows() – трансформація при розшифруванні яка є зворотною стосовно ShiftRows().

– InvSubBytes() – трансформація при розшифруванні яка є зворотною стосовно SubBytes().

– InvMixColumns() – трансформація при розшифруванні яка є зворотною стосовно MixColumns().

– RotWord() – функція, що використовується в процедурі Key Expansion, що бере 4-х байтне слово й робить над ним циклічну перестановку.

– SubWord() – функція, використовувана в процедурі Key Expansion, що бере на вході 4-х байтне слово й застосовуючи S-box до кожного із чотирьох байтів видає вихідне слово.

– Block – послідовність біт, з яких складається input, output, State і Round Key. Також під Block можна розуміти послідовність байт.

– Ciphertext – вихідні дані алгоритму шифрування.

– S-box – нелінійна таблиця замін, що використовується в декількох трансформаціях заміни байт і в процедурі Key Expansion для взаємнооднозначної заміни значення байта.

– Nb – число стовпців(32-ух бітних слів), що становлять State. Для, AES Nb = 4.

– Nk – число 32-ух бітних слів, що становлять шифроключ. Для AES, Nk = 4,6, або 8.

– Nr – число раундів, що є функцією Nk і Nb. Для AES, Nr = 10, 12, 14.

– Rcon[] – масив, що складається з бітів 32-х розрядного слова і є постійним для даного раунду.

Блок менеджеру паролів

Надає можливості не тільки для простого збереження паролів, але й для повноцінної роботи з ними. Програма підтримує роботу з декількома аккаунтами, і працювати з нею можуть трохи користувачів. При цьому бази даних кожного користувача шифруються.

Додаткові можливості:

– Система пошуку по базі даних.

– Підтримка макросів.

– Можливість резервного копіювання бази даних.

– Можливість швидкого перемикання між користувачами.

– Швидкий доступ до часто використовуваних функцій.

– Генератор паролів.

– Можливість друку паролів.

Допомога

Блок призначений про надання допомоги по роботі з системою, а також для надання інформації про розробників системи, версію та дату випуску.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

Блок журналювання подій

Призначений для запису у журнал усіх подій, які відбуваються у системі. Журнал дій користувачів містить форму для запуску архівації журналу. Форма архівації являє собою кнопку "Очистити журнал" і поле з датою "по:". Дату можна встановлювати будь-яку, але не раніше, ніж поточна дата мінус 1 місяць, щоб у системі завжди зберігалися дані про дії користувачів як мінімум за місяць.

Після натискання кнопки "Очистити журнал" у Системі генерується текстовий файл із архівом журналу за обраний період. Файл зберігається в зашифрованому виді, а посилання на цей файл показуються адміністраторові. Після створення файлу запису журналу за обраний період віддаляються з бази даних. У випадку помилки при створенні або збереженні файлу, записи не видаляються.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.11. З нього видно, що процеси у системі взаємодіють наступним чином. Спершу завантажується процес виведення головного вікна програми.

Він взаємодіє з наступними процесами:

– Процес дешифрування.

– Процес шифрування.

Процес дешифрування взаємодіє з наступними процесами:

– Процес відкриття файлу для дешифрування, який, у свою чергу, взаємодіє з процесом виведення інформації про файл до дешифрування.

– Процес виведення повідомлення про завершення дешифрування.

– Процес виведення часу дешифрування.

– Процес збереження дешифрованого файлу.

– Процес введення ключа шифрування.

Процес шифрування взаємодіє з наступними процесами:

– Процес відкриття файлу для шифрування, який, у свою чергу, взаємодіє з процесом виведення інформації про файл до шифрування.

– Процес виведення повідомлення про завершення шифрування.

– Процес виведення часу шифрування.

– Процес збереження зашифрованого файлу.

– Процес введення ключа шифрування.

Рисунок 3.11 – Діаграма взаємодії процесів

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього користувач обирає, яку дію йому виконувати:

– Шифрувати файл.

– Розшифровувати зашифрований файл.

Якщо користувач обирає шифрувати файл, тоді виконуються наступні ітерації:

– Вибір файлу для шифрування.

– Виведення назви та розміру файлу.

– Введення ключа шифрування.

– Виконання підпрограми шифрування файлу.

– Виведення повідомлення про завершення шифрування.

– Виведення часу шифрування в мілісекундах.

– Збереження зашифрованого файлу.

Якщо користувач обирає дешифрувати файл, тоді виконуються наступні ітерації:

– Вибір файлу для дешифрування.

– Виведення назви та розміру файлу.

– Введення ключа шифрування.

– Виконання підпрограми дешифрування файлу.

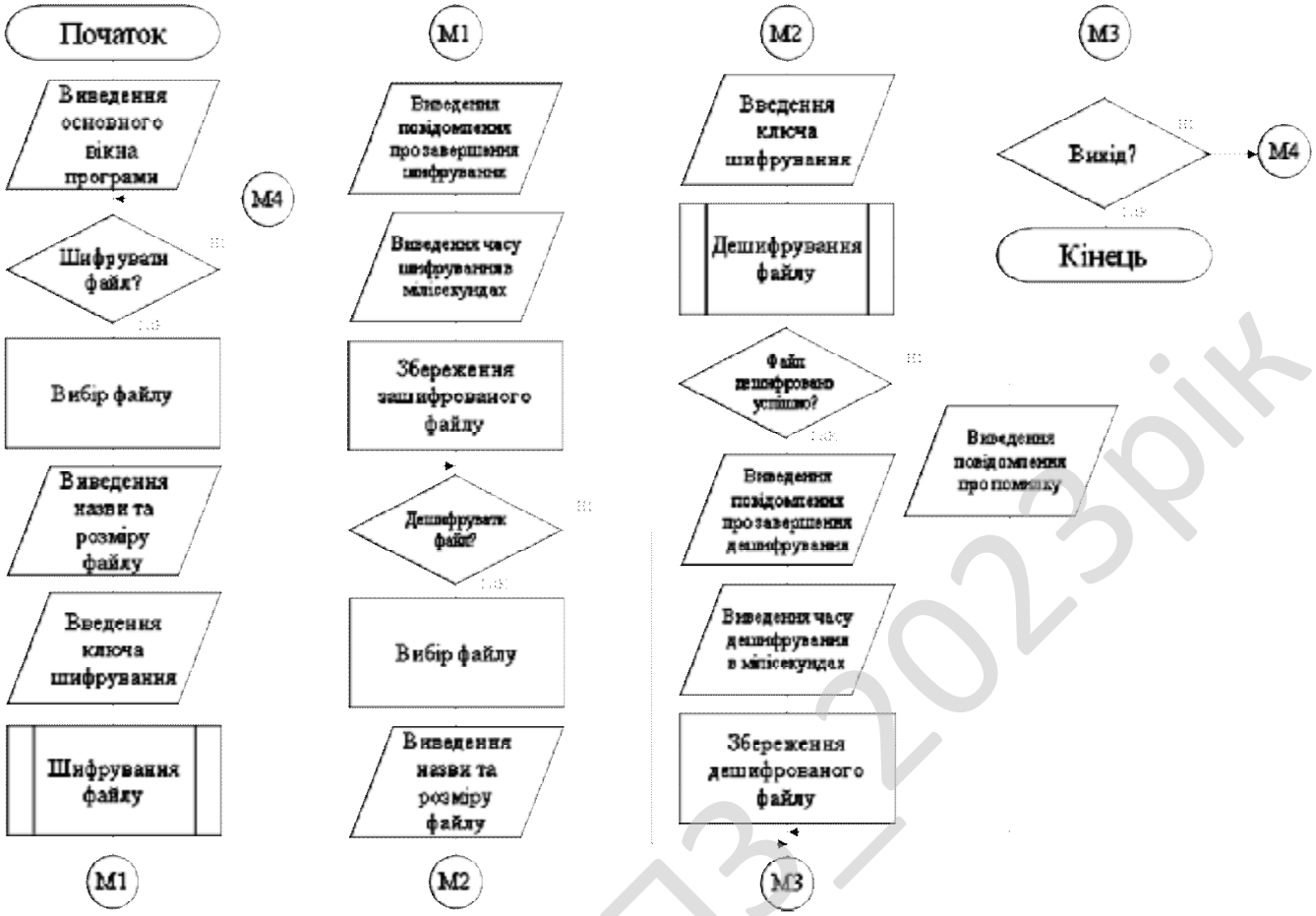

Рисунок 4.1 – Блок-схема роботи основної програми

Після цього програма визначає файл дешифровано успішно, або ні.

Якщо файл дешифровано успішно, тоді виконуються наступні дії:

– Виведення повідомлення про завершення дешифрування.

– Виведення часу дешифрування в мілісекундах.

– Збереження дешифрованого файлу.

У противному випадку відбувається виведення повідомлення про помилку.

На рисунку 4.2 зображена блок-схема роботи підпрограми шифрування/дешифрування алгоритмом AES з використанням спеціальних інструкцій Intel Core i9-10900K. Підпрограма працює наступним чином.

Спершу визначається шифрувати, або дешифрувати файл.

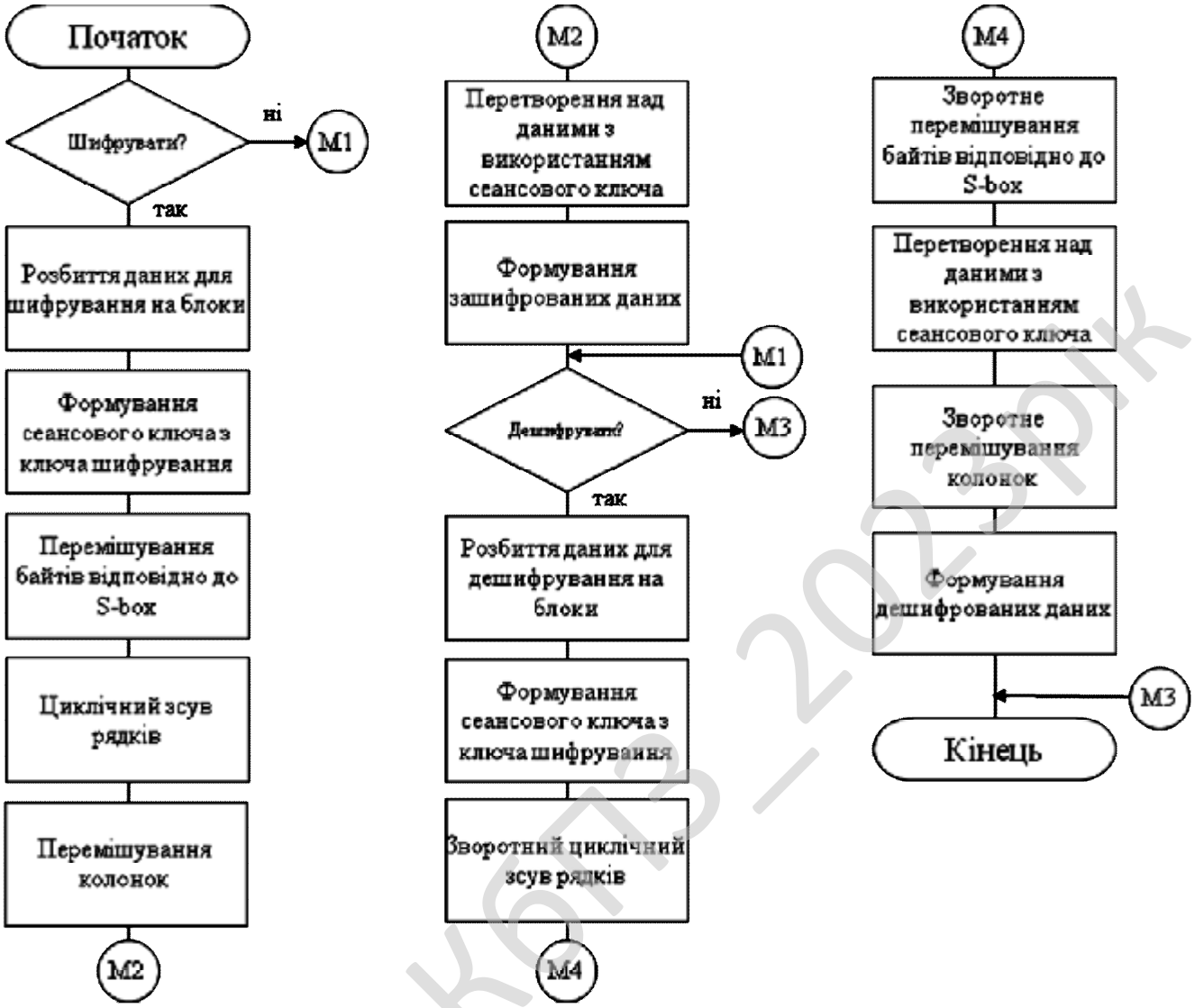

Рисунок 4.2 – Блок-схема роботи підпрограми шифрування/дешифрування алгоритмом AES з використанням спеціальних інструкцій Intel Core i9-10900K

Якщо потрібно шифрувати, тоді виконуються наступні дії:

– Розбиття даних для шифрування на блоки.

– Формування сеансового ключа з ключа шифрування.

– Перемішування байтів відповідно до S-box.

– Циклічний зсув рядків.

– Перемішування колонок.

– Перетворення над даними з використанням сеансового ключа.

– Формування зашифрованих даних.

Якщо потрібно дешифрувати, тоді виконуються наступні дії:

– Розбиття даних для дешифрування на блоки.

– Формування сеансового ключа з ключа шифрування.

– Зворотний циклічний зсув рядків.

– Зворотне перемішування байтів відповідно до S-box.

– Перетворення над даними з використанням сеансового ключа.

– Зворотне перемішування колонок.

– Формування дешифрованих даних.

На цьому підпрограма закінчує свою роботу.

Наведемо код, який призначений для дешифрування.

```
function DecryptData(Data: string; AKey: AnsiString; AIv: AnsiString):
string;
var
  key, iv, src, dest: TBytes;
  cipher: TDCP_rijndael;
  slen, pad: integer;
begin
  //key := Base64DecodeBytes(TEncoding.UTF8.GetBytes(AKey));
  //iv := Base64DecodeBytes(TEncoding.UTF8.GetBytes(AIv));
  key := TEncoding.ASCII.GetBytes(AKey);
  iv := TEncoding.ASCII.GetBytes(AIv);
  src := Base64DecodeBytes(TEncoding.UTF8.GetBytes(Data));
  cipher := TDCP_rijndael.Create(nil);
  try
    cipher.CipherMode := cmCBC;
    slen := Length(src);
    SetLength(dest, slen);
    cipher.Init(key[0], 256, @iv[0]);    cipher.Decrypt(src[0], dest[0],
slen);
    pad := dest[slen - 1];
    SetLength(dest, slen - pad);
    result := TEncoding.Default.GetString(dest);
  finally
    cipher.Free;
  end;
end;
```

Наведемо код, який призначений для шифрування.

```
function EncryptData(Data: string; AKey: AnsiString; AIv: AnsiString):
string;
var
  cipher: TDCP_rijndael;
  key, iv, src, dest, b64: TBytes;
  index, slen, bsize, pad: integer;
begin
  //key := Base64DecodeBytes(TEncoding.UTF8.GetBytes(AKey));
  //iv := Base64DecodeBytes(TEncoding.UTF8.GetBytes(AIv));
  key := TEncoding.ASCII.GetBytes(AKey);
  iv := TEncoding.ASCII.GetBytes(AIv);
  src := TEncoding.UTF8.GetBytes(Data);
  cipher := TDCP_rijndael.Create(nil);
  try
    cipher.CipherMode := cmCBC;
    slen := Length(src);
    bsize := (cipher.BlockSize div 8);
    pad := bsize - (slen mod bsize);
    Inc(slen, pad);
    SetLength(src, slen);
    for index := pad downto 1 do
    begin
      src[slen - index] := pad;
    end;
    SetLength(dest, slen);
    cipher.Init(key[0], 256, @iv[0]); // DCP uses key size in BITS not
BYTES
    cipher.Encrypt(src[0], dest[0], slen);
    b64 := Base64EncodeBytes(dest);
    result := TEncoding.Default.GetString(b64);
  finally
    cipher.Free;
  end;
end;
```

AES розшифровується як "Advanced Encryption Standard" – це найбільш популярний стандарт симетричного шифрування у світі IT. Стандарт працює із блоками розміром 128 біт і підтримує 128-, 192- або 256-бітні ключі (AES-128, AES-192 і AES-256). Багато утиліт шифрування, та ж TrueCrypt, підтримали

алгоритм AES на самому початку його існування. Але найбільший фактор успіху AES, звичайно, полягає в його прийнятті урядом США в 2002 році, при цьому в 2003 році він був прийнятий як стандарт для захисту секретних даних.

Шифрування даних за допомогою AES

Шифрування AES базується на системі підстановок з перестановкою, тобто над даними проводиться серія математичних операцій, щоб створити значно модифікований масив даних (зашифрований). Як вихідна інформація виступає текст, а ключ відповідає за виконання математичних операцій. Операції можуть бути як зовсім простими, наприклад, зрушення бітів або XOR, так і більше складними. Один прохід можна легко розшифрувати, тому всі сучасні алгоритми шифрування побудовані на декількох проходах. У випадку AES це 10, 12 або 14 проходів для AES-128, AES-192 або AES-256. До речі, ключі AES проходять таку ж процедуру, що й користувальницькі дані, тобто вони являють собою що змінюється раундовий ключ.

Процес працює з масивами 4x4 з одиночних байтів, також називаних боксами: S-box використовуються для підстановок, P-box – для перестановок. Підстановки й перестановки виконуються на різних етапах: підстановки працюють усередині так званих боксів, а перестановки міняють інформацію між боксами. S-box працює по складному принципі, тобто навіть якщо єдиний вхідний біт буде мінятися, то це вплине на декілька вихідних бітів, тобто властивості кожного вихідного біта залежать від кожного вхідного біта.

Використання декількох проходів забезпечує гарний рівень шифрування, при цьому необхідно відповідати критеріям розсіювання (diffusion) і заплутування (confusion). Розсіювання виконується через каскадну комбінацію трансформацій S-box і P-box: при зміні тільки одного біта у вхідному тексті S-box буде модифікувати вихід декількох біт, а P-box буде псевдовипадково поширювати цей ефект по декількох S-box. Коли ми говоримо про те, що мінімальна зміна на вході дає максимальна зміна на виході, ми говоримо про ефект сніжної грудки.

Надійність шифрування AES

Останнім часом іде чимало обговорень так званих взламів, які обходять необхідність запуску розширеного пошуку методом грубої сили для знаходження правильного ключа розшифровки. Технології, такі як атаки XSL і атаки related-key обговорюються досить інтенсивно – але успіх невеликий. Єдиний працюючий спосіб злому шифрування AES полягає в так званій атаці побічного каналу (side-channel). Для її здійснення атака повинна відбуватися тільки на host-системі, на якій виконується шифрування AES, і при цьому вам необхідно знайти спосіб одержання інформації про синхронізацію кеша. У такому випадку можна відстежити число тактів комп'ютера до завершення процесу шифрування.

Звичайно, все це не так легко, оскільки вам потрібен доступ до комп'ютера, причому досить повний доступ для аналізу шифрування й права на виконання коду. Тепер вам напевно зрозуміло, чому "діри" у системі безпеки, які дозволяють зловмисникові одержати такі права, нехай навіть вони звучать зовсім абсурдно, необхідно закривати якнайшвидше. Але якщо ви одержите доступ до цільового комп'ютера, то добування ключа AES – справа часу, тобто вже не трудомістке завдання для суперкомп'ютерів, що вимагає величезних обчислювальних ресурсів.

AES усередині Intel

На даний момент інтегровані в CPU інструкції AES починають мати сенс – незалежно від можливих переваг по продуктивності. З погляду безпеки процесор може обробляти інструкції AES в інкапсульованому виді, тобто йому не потрібні які-небудь таблиці перетворення, необхідні для атаки методом побічного каналу.

Процесори насправді знаменують собою зміну поколінь, оскільки при цьому не тільки відбувається перехід на наступний техпроцес (32 нм у порівнянні з 45 нм), але перед нами й перше покоління CPU з підтримкою декількох інструкцій, що прискорюють шифрування. Intel згадує добавку як AES New Instructions. Вони складаються із чотирьох інструкцій для шифрування AES (AESENC, AESENCLAST) і розшифровки (AESDEC, AESDECLAST) плюс ще дві

інструкції для роботи із ключем AES (AESIMC, AESKEYGENASSIST). Як і раніше, інструкції відносяться до SIMD, тобто до типу "одна інструкція багато даних" (Single Instruction Multiple Data). Підтримуються всі три ключі AES (128, 192 і 256 бітів з 10, 12 і 14 проходами підстановки й перестановки).

Оскільки всі інструкції AES мають фіксовану затримку, що не залежить від даних, тобто час фіксоване й доступ до пам'яті не потрібно. Крім того, модель програмування така ж, як і у випадку інших інструкцій SSE з первісного стандарту SSE4. Таким чином, всі операційні системи, які підтримують роботу з SSE, зможуть використовувати й інструкції AES New Instructions.

Будьте обережні при виборі процесора із прискоренням AES, оскільки сьогодні лише деякі моделі підтримують нові інструкції. 32-нм процесори Core i3 на Clarkdale не підтримують інструкції, а дводерна лінійка Core i9-10900K-600 – підтримує. У мобільних процесорів ситуація ледве більше складна: якщо мобільні процесори Core i3 теж не підтримують прискорення AES, те процесори лінійки Core i9-10900K-500 уже підтримують. Однак є одна модель Core i9-10900K-400, позбавлена такої підтримки. Усе було б набагато простіше, якби Intel додала підтримку нових інструкцій в усі моделі.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою CRYPTON – алгоритм симетричного блочного шифрування (розмір блоку 128 біт, ключ довжиною до 256 біт), розроблений південнокорейським криптологом Чьо Лім Хун з південнокорейської компанії Future Systems, яка з кінця 1980-х років працює на ринку забезпечення мереж і захисту інформації. Алгоритм був розроблений в 1998 році в якості шифру – учасника конкурсу AES. Як зізнавався автор, конструкція алгоритму спирається на алгоритм SQUARE[1]. В алгоритмі Crypton немає традиційних для блочних шифрів мережі Фейстеля. Основу даного шифру становить так звана SP-мережа (повторювана циклова

функція, що складається із замін-перестановок, орієнтована на розпаралелену нелінійну обробку всього блоку даних). Крім високої швидкості, перевагами таких алгоритмів є полегшення дослідження стійкості шифру до методів диференціального та лінійного криптоаналізу, що є на сьогодні основними інструментами розтину блочних шифрів. На конкурс AES була представлена версія алгоритму Crypton v0.5. Однак, як казав Чьо Лім Хун, йому не вистачало часу для розробки повної версії. І вже на першому етапі конкурсу AES в ході аналізу алгоритмів, версія Crypton v0.5 була замінена на версію Crypton v1.0. Відмінність нової версії від первинної полягала в зміні таблиці замін та в модифікації процесу розширення ключа.

Як і інші учасники конкурсу AES, Crypton призначений для шифрування 128-бітових блоків даних[2]. При шифруванні використовуються ключі шифрування для декількох фіксованих розмірів – від 0 до 256 біт з кратністю 8 бітів. Структура алгоритму Crypton – структура «Квадрата» – багато в чому схожа на структуру алгоритму Square, створеного в 1997 році. Криптографічні перетворення для алгоритмів з даною структурою можуть бути виконані як для цілих рядків і стовпців масиву, так і над окремими його байтами. (Варто зазначити, що алгоритм Square був розроблений авторами майбутнього переможця конкурсу AES – авторами алгоритму Rijndael – Вінсентом Ріджменом і Джоан Дейменом.)

Шифрування

Алгоритм Crypton являє 128-бітовий блок шифруємих даних у вигляді байтового масиву 4×4, над якими в процесі шифрування проводиться кілька раундів перетворень. У кожному раунді передбачається послідовне виконання наступних операцій:

– Таблична заміна γ;

– Лінійне перетворення π;

– Байтова перестановка τ;

– Операція σ.

Таблична заміна γ

Алгоритм Crypton використовує 4 таблиці замін. Кожна з яких заміщає 8-бітне вхідне значення на вихідне такого ж розміру.

Лінійне перетворення π

Тут використовується 4 спеціальні константи. Ці константи об'єднані в маскуючі послідовності

Байтова перестановка τ

Дана перестановка перетворює найпростішим чином рядок даних у стовпець.

Операція σ

Дана операція є побітовим складанням всього масиву даних з ключем раунду. Зауважимо, саме 12 раундів шифрування рекомендується автором алгоритму Чьо Хун Лімом, проте сувора кількість раундів не встановлена.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ КІБЕРБЕЗПЕКИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

На рисунку 5.1 зображено інтерфейс користувача у вигляді головного вікна програми. З рисунку видно, що головне вікно складається з наступних блоків:

– Блок меню.

– Блок кнопок швидкого доступу до елементів програми.

– Блок виведення даних.

Блок меню складається з наступних елементів:

– Файл.

– Шифрування.

– Дешифрування.

– Довідка.

Блок кнопок швидкого доступу до елементів програми складається з наступних елементів:

– Перегляд файлу.

– Ключ з файлу.

– Шифрування.

– Дешифрування.

Блок виведення даних складається з наступних елементів:

– Розмір файлу.

– Статус.

– Час шифрування.

– Час дешифрування.

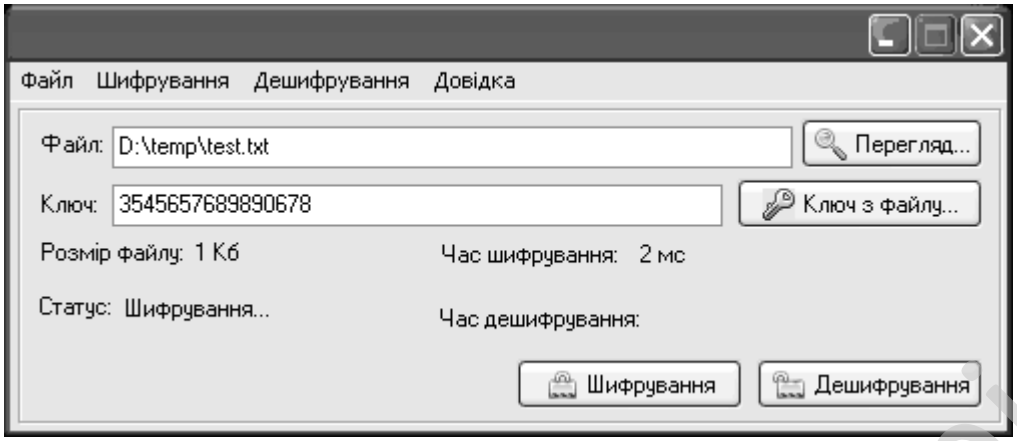

Рисунок 5.1 – Головне вікно програми – шифрування файлу

На рисунках 5.2-5.3 наведено скріншоти файлу до шифрування й після шифрування.

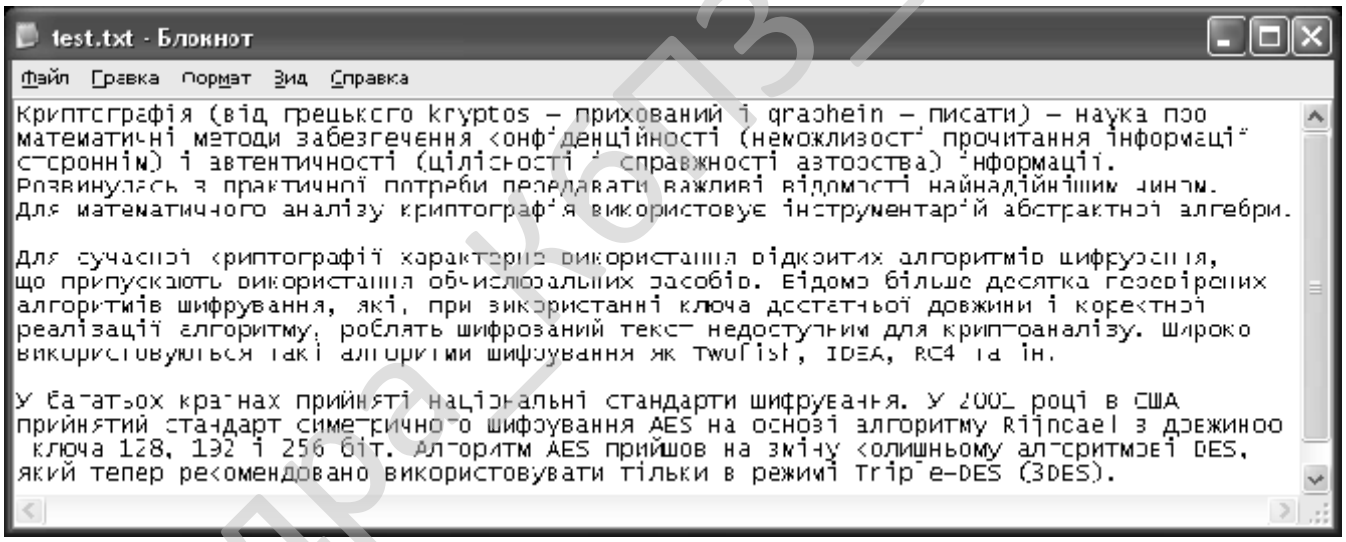

Рисунок 5.2 – Файл до шифрування

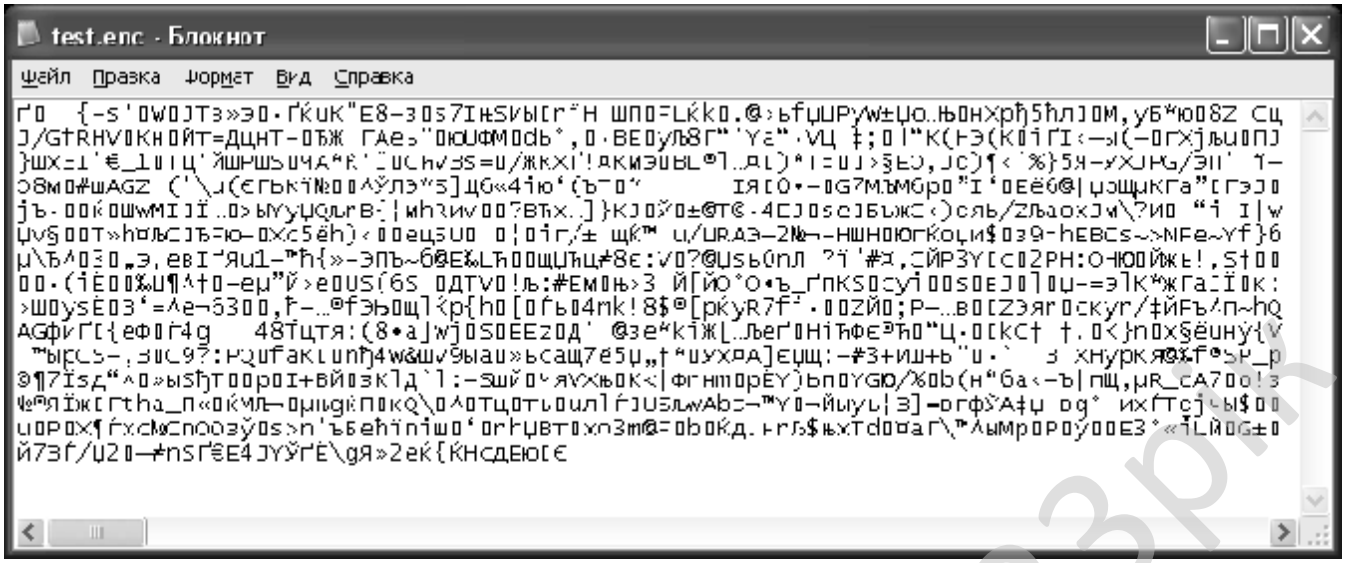

Рисунок 5.3 – Зашифрований файл

На рисунках 5.4-5.5 наведені скріншоти дешифрування файлу.

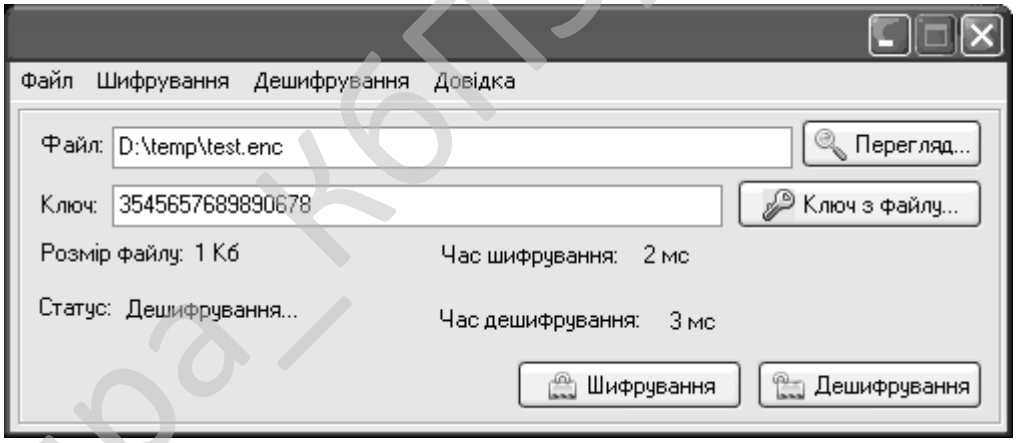

Рисунок 5.4 – Головне вікно програми – дешифрування файлу

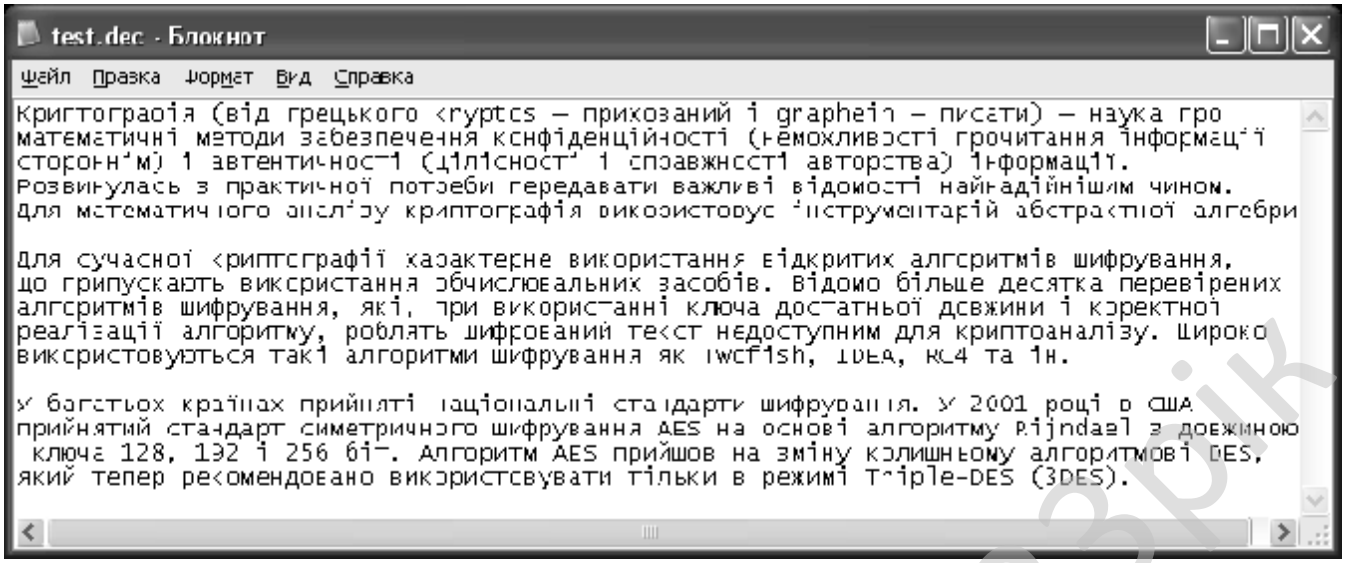

Рисунок 5.5 – Дешифрований файл

На рисунку 5.6 заведено вікно довідки, з якого можливо отримати такі дані, як:

– Розробник програмного забезпечення.

– Керівник проекту.

– Тема та місце виконання проекту

Рисунок 5.6 – Довідка

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

– Досліджена система для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

– На основі отриманих результатів досліджень створена програмна реалізація системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Delphi 10. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для

системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K. Це дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи кібербезпеки й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи кібербезпеки Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм CRYPTON.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Smirnov, O., Neskorodieva, T., Fedorov, E., Rudakov, K., Neskorodieva, A. «Method Detection Audit Data Anomalies on Basis Restricted Cauchy Machine» *CEUR Workshop Proceedings,* Volume 3187, 2022, pp. 1-12. **(Scopus).**

2. Smirnov O., Smirnova T., Anas M. Al-Oraiqat, Drieiev O., Polishchuk L., Sheroz Khan, Yassin M. Y. Hasan, Aladdein M. Amro, Hazim S. AlRawashdeh «Method for Determining Treated Metal Surface Quality Using Computer Vision Technology». *Sensors (Basel, Switzerland)* Volume 22, Issue 16, 6223, 2022. (**Scopus**).

3. Smirnov, O., Lakhno, V., Akhmetov, B., Chubaievskyi, V., Khorolska, K., Bebeshko, B. «Selection of a Rational Composition of İnformation Protection Means Using a Genetic Algorithm». *In: Rajakumar, G., Du, KL., Vuppalapati, C., Beligiannis, G.N. (eds) Intelligent Communication Technologies and Virtual Mobile Networks. Lecture Notes on Data Engineering and Communications Technologies*, vol 131. 2023. **Springer**, Singapore. pp. 21-34. **(Scopus).**

4. Smirnov O.A., Al-Oraiqat A.M., Ulichev O.S., Meleshko Ye.V., Al-Rawashdeh H.S., Polishchuk L.I. «Modeling strategies for information influence dissemination in social networks». *Journal of Ambient Intelligence and Humanized Computing* Volume 13, Issue 5. **Springer**, Cham. 2022, pp. 2463-2477. **(Scopus).**

5. Smirnov O., Kuznetsov A., Kryvinska N., Kiian A., Kuznetsova K. «Full Non-Binary Constant-Weight Codes». *SN Computer Science,* Vol 2, 337, 2021. https://doi.org/10.1007/s42979-021-00739-w **(Scopus).**

6. Smirnov O., Kovalenko O., Kovalenko A., Kavun S. «Quantitative Risk Assessment Method Development in the Context of the SDLC-model». *2021 IEEE 8th International Conference on Problems of Infocommunications, Science and Technology (PIC S&T),* 2021, pp. 203-208, doi: 10.1109/PICST54195.2021.9772143 **(Scopus).**

7. Smirnov O., Neskorodieva T., Fedorov E., Rymar P. «Neural Network Modeling Method of Transformations Data of Audit Production with Returnable Waste». *CEUR Workshop Proceedings* Volume 3101, 2021, Pages 192-207. **(Scopus).**

8. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova K. «Data hiding scheme based on spread sequence addressing». *CEUR Workshop Proceedings* Volume 2805, 2020, Pages *44-58.* **(Scopus)**.

9. Smirnov, O., Kuznetsov, A., Potii, O., Poluyanenko, N., Stelnyk, I., Mialkovsky, D. «Combining and filtering functions in the framework of nonlinear-feedback shift register». *International Journal of Computing*; 2020, Volume 19, Issue 2 – Research Institute for Intelligent Computer Systems – 2020. – P. 247-256. **(Scopus)**.

10. Smirnov O., Kuznetsov A., Kiian A., Kuznetsova T. «Non-binary constant weight coding technique». *CEUR Workshop Proceedings.* Volume 2740, 2020, Pages 102-114. **(Scopus)**.

11. Smirnov O.A., Alimseitova Zh., Adranova A., Akhmetov B., Lakhno V., Zhilkishbayeva G. «Models and algorithms for ensuring functional stability and cybersecurity of virtual cloud resources». *Journal of theoretical and applied information technology* Vol.98. No 21, 2020, P. 3334-3346. **(Scopus)**.

12. Smirnov O., Kuznetsov A., Arischenko A., Chepurko I., Onikiychuk A., Kuznetsova T. «Pseudorandom sequences for spread spectrum image steganography». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 122-131. **(Scopus)**.

13. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New technique for data hiding in cover images using adaptively generated pseudorandom sequences». *CEUR Workshop Proceedings* Volume 2654, 2020, Pages 1-14. **(Scopus)**.

14. Smirnov O., Lutsenko M., Kuznetsov A., Kiian A., Kuznetsova T., «Biometric cryptosystems: overview, state-of-the-art and perspective directions». *Lecture Notes in Networks and Systems*, vol 152. **Springer**, Cham. 2021, pp 66-84. **(Scopus)**.

15. Smirnov O., Kuznetsov A., Pushkar'ov A., Serhiienko R., Babenko V., Kuznetsova T., «Representation of Cascade Codes in the Frequency Domain». In: Radivilova T., Ageyev D., Kryvinska N. (eds) Data-Centric Business and Applications. Lecture Notes on Data Engineering and Communications Technologies, vol 48. **Springer**, Cham. 2021. pp 557-587. **(Scopus)**.

16. Smirnov, O., Markovets, O. Vovk, N., Turchyn, Y., «Model of informational support for social network administrators' content creation». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 125-136. **(Scopus)**.

17. Smirnov, O., Drieieva, H., Drieiev, O., Polishchuk, Y., Brzhanov, R., Aleksander, M. «Method of fractal traffic generation by a model of generator on the graph». *CEUR Workshop Proceedings* Volume 2616, 2020, Pages 366-379. **(Scopus)**.

18. Smirnov, O., Shekhanin, K., Kuznetsov, A., Krasnobayev, V. «Detecting Hidden Information in FAT». *International Journal of Computer Network and Information Security (IJCNIS)*. Vol. 12, No. 3, 2020. PP.33-43. **(Scopus)**.

19. Smirnov, O., Drieieva, H., Drieiev, O., Simakhin, V., Bondar, S., Odarchenko, R. «Managing multifractal properties of the binary sequence generated with the Markov chains», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 633-645. **(Scopus)**.

20. Smirnov, O., Kuznetsov, A., Gorbacheva, L., Babenko, V., «Hiding data in images using a pseudo-random sequence», *CEUR Workshop Proceedings* Volume 2608, 2020, Pages 646-660., **(Scopus)**.

21. Smirnov, O., Kuznetsov, A., Kolovanova, I., Kuznetsova, T., «Noise immunity of the algebraic geometric codes». International Journal of Computing; 2019, Volume 18, Issue 4 – Research Institute for Intelligent Computer Systems – 2019. – P. 393-407. (**Scopus)**.

22. Smirnov, O., Ulichev, O., Meleshko, Y., Khokh, V., Goncharenko, I. «Method of Choosing Objects for Informational Influence in Social Networks during Information Campaign Based on the Analytic Hierarchy Process». *CEUR Workshop Proceedings*, Vol 2588, P. 215-227, 2019. **(Scopus)**.

23. Smirnov, O., Krasnobayev, V., Yanko, A., Kuznetsova, T. «Methods of nulling numbers in the system of residual classes». *CEUR Workshop Proceedings*, Vol 2588, P. 90-106, 2019. **(Scopus)**.

24. Smirnov, O., Kuznetsov, A., Kovalchuk, D., Pastukhov, M., Kuznetsova, K., Prokopovych-Tkachenko, D., «Discrete Signals with Special Correlation Properties», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 618-629. **(Scopus)**.

25. Smirnov, O., Kuznetsov, A., Kiian, A., Kuznetsova, K., Ivko, T., Prokopovych-Tkachenko, D., «Soft Decoding Based on Ordered Subsets of Verification Equations of Turbo-Productive Codes», *CEUR Workshop Proceedings* Volume 2353, *CEUR Workshop Proceedings* 2019, Pages 873-884. **(Scopus)**.

26. Smirnov, O., Kuznetsov, A., Prokopovych-Tkachenko, D. «Hiding Data in Images Using a Pseudo-Random Sequence». *ISCI'2020: Information Security in Critical Infrastructures.* ***Collective monograph***. Edited by Ivan D. Gorbenko, Victor A. Krasnobayev and Alexandr A. Kuznetsov. ASC Academic Publishing, USA, 2020. pp. 46-59. – ISBN: 978-1-7362833-0-1 (Hardback), ISBN: 978-1-7362833-1-8 (Ebook).

27. Smirnov, O., Kuznetsov, A., Shekhanin, K., Chepurko, I. Detecting Hidden Information in FAT. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures**. Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 412-429. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

28. Smirnov, O., Kuznetsov, A., Kuznetsova., K. Synthesis of Discrete Signals with Improved Correlation Properties. Монографія: In.: ISCI'2019: Information Security in Critical Infrastructures. **Collective monograph**. Edited by Ivan D. Gorbenko and Alexandr A. Kuznetsov, ASC Academic Publishing, USA, 2019, pp. 281-299. – ISBN: 978-0-9989826-8-7 (Hardback), ISBN: 978-0-9989826-9-4 (Ebook).

29. О.А. Смірнов, П.С. Усік, «Дослідження перспектив використання технологічних рішень в мережах 5G» у *Кібербезпека та інформаційні технології: монографія.* – Х. : ТОВ «ДІСА ПЛЮС», 2020.С. 122-135.

30. Смірнов О.А., Дрєєва Г.М., «Метод генерування фрактального трафіку за допомогою моделі генератора на графі» у Інформаційна безпека та інформаційні технології: монографія / за заг. ред. В. С. Пономаренка. – Х. : Вид. Рожко С.Г. 2019. С. 123-139.

31. Смирнов А.А., Коваленко А.В. Комплекс математических моделей технологии тестирования WEB-приложений. Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

32. Смирнов А.А., Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения. Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

33. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Смірнов С.А., Поліщук Л.І., «Дослідження стійкості до диференціального криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку,* 2022, № 3(69). С. 93-98. 2022.

34. Смірнов О.А., Смірнова Т.В., Якименко Н.М., Поліщук Л.І., Смірнов С.А. «Дослідження статистичної стійкості та швидкісних характеристик запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Вісник Хмельницького національного університету. Серія: «Технічні науки»,* № 2 (307). С. 46-52. 2022.

35. Смірнов О.А., Смірнова Т.В., Константинова Л.В., Смірнов С.А., Якименко Н.М., «Дослідження стійкості до лінійного криптоаналізу запропонованої функції гешування удосконаленого модуля криптографічного захисту в інформаційно-комунікаційних системах» *Системи управління, навігації та зв'язку*, 2022, № 1(67). С. 84-89.

36. Смірнов О.А., Смірнова Т.В., Буравченко К.О., Кравченко С.С., Горбов В.О., «Хмарна система підтримки прийняття рішень технологічного

процесу відновлення поверхонь конструкцій і деталей машин». *Сучасні інформаційні системи*. 2021. Т. 5, № 4. С. 79-95

37. Смирнов А., Кузнецов А., Кузнецова Т. «Шумоподобные дискретные сигналы для асинхронных систем кодового разделения радиоканалов». *Радиотехника*, № 2(205), 175–183. 2021.

38. Smirnov O., Kuznetsov A., Kovalchuk D., Kuznetsova T. «New Technique for Hiding Data in Cover Images Using Adaptively Generated Pseudorandom Sequences». *CEUR Workshop Proceedings* Volume 2732, 2020, Pages 214-227.

39. Смірнов, О.А., Полігенько О.О., Одарченко Р.С., Терещенко Л.Ю.Усік П.С., «Інформаційна технологія та програмне забезпечення для підвищення ефективності планування підсистеми базових станцій стільникового зв'язку». *Проблеми телекомунікацій*. № 1(26). С. 83-96. 2020.

40. Смирнов А.А, Кузнецов А.А., Киян А.С., Кузнецова Е.А. «Сокрытие данных на основе адресации шумоподобных сигналов». *Всеукраїнський міжвідомчий науково-технічний збірник "Радіотехніка"* – Харьків: ХНУРЕ. – 2020. – Вип. 203. – С. 38-49.

41. Смирнов А.А., Дудан А.В., Смирнова Т.В. «Формализация структуры технологического процесса электродугового напыления». *Сборник научных трудов «Актуальные вопросы машиноведения».* Объединенный институт машиностроения Национальной Академии Наук Беларуси. №9. С. 308-312, 2020.

42. Смірнов О.А., Усік П.С., Миронець І.В., Буравченко К.О., Якименко Н.М. «Метод підвищення ефективності розподіленої обробки даних у комп'ютерних системах операторів стільникового зв'язку» *Вісник Черкаського державного технологічного університету. Технічні науки.* №4. С. 103-110. 2020.

43. О.А.Смірнов, Т.В.Смірнова, Л.І. Поліщук, К.О. Буравченко, А.О.Макевнін, «Дослідження хмарних технологій як сервісів», *Кібербезпека: освіта, наука, техніка*. № 3(7). С. 43-62. 2020.

44. А.А. Смирнов, Т.В. Смирнова, А.Н. Дреев, А.В. Дудан. «Оптимизация технологического процесса восстановления и упрочнения поверхностей с заданными характеристиками в виде облачного сервиса». Вестник Полоцкого государственного университета. Серия В, Промышленность. Прикладные науки. Республика Беларусь - 2020. - № 3. - С. 50-61.

45. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., Смірнова Т.В. «Фрактальний аналіз генератора самоподібного трафіку на основі ланцюга Маркова». *Центральноукраїнський науковий вісник. Технічні науки.* № 2(33). с. 161-172, 2019.

46. О.А. Смірнов, Т.В. Смірнова, О.М. Дрєєв, Є.К. Солових, «Методи оптимізації технологічних процесів відновлення сталевих покриттів», *Shipbuilding & marine infrastructure / Суднобудування і морська інфраструктура* № 1 (11). с. 48-57, 2019.

47. Смірнов О.А., Дрєєва Г.М., Дрєєв О.М., «Побудова хмарних інформаційних технологій оптимізації технологічного процесу відновлення та зміцнення поверхонь деталей». Центральноукраїнський науковий вісник. Технічні науки. № 1(32). с. 184-194, 2019.

48. Смірнов О.А., Смірнов С.А., Поліщук Л.І., Смірнова Т.В., Конопліцька-Слободенюк О.К. Метод формування антивірусного захисту даних з використанням безпечної маршрутизації метаданих. Кібербезпека: освіта, наука, техніка. – Том 3 № 3. – Київ: КУ ім. Бориса Грінченка. – 2019. – С. 63-87.

49. Смирнов А.А., Лысенко И.А., Информационная технология проектирования тестовых наборов на основе требований к программному обеспечению, Системи управління, навігації та зв'язку. – Випуск 4 (44). – Полтава: ПолтНТУ. – 2017. – С. 112-115.

50. Смірнов О.А., Мелешко Є.В., Хох В.Д., Дослідження методів аудиту систем управління інформаційною безпекою, Системи управління, навігації та зв'язку. – Випуск 1 (41). – Полтава: ПолтНТУ. – 2017. – С. 38-42.

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 12-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи кібербезпеки з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Delphi 10.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 85 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 8.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Смірнов С.А.

Програмне забезпечення системи кібербезпеки для забезпечення захисту інформації з використанням спеціальних інструкцій Intel Core i9-10900K

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 53

Літера: РП

Кропивницький – 2023 року

Файл El_Intel_Core_i9_10900K_AES_.pas – шифрування/дешифрування алгоритмом _Intel_Core_i9_10900K_AES_

```
unit El_Intel_Core_i9_10900K_AES_;

interface

uses
  Classes, SysUtils;

type
  E_Intel_Core_i9_10900K_AES_Error = class(Exception);

  PInteger  = ^Integer;

  T_Intel_Core_i9_10900K_AES_Buffer = array [0..15] of byte;
  T_Intel_Core_i9_10900K_AES_Key128 = array [0..15] of byte;
  T_Intel_Core_i9_10900K_AES_Key192 = array [0..23] of byte;
  T_Intel_Core_i9_10900K_AES_Key256 = array [0..31] of byte;
  T_Intel_Core_i9_10900K_AES_ExpandedKey128 = array [0..43] of longword;
  T_Intel_Core_i9_10900K_AES_ExpandedKey192 = array [0..53] of longword;
  T_Intel_Core_i9_10900K_AES_ExpandedKey256 = array [0..63] of longword;

  P_Intel_Core_i9_10900K_AES_Buffer =^T_Intel_Core_i9_10900K_AES_Buffer;
  P_Intel_Core_i9_10900K_AES_Key128 =^T_Intel_Core_i9_10900K_AES_Key128;
  P_Intel_Core_i9_10900K_AES_Key192 =^T_Intel_Core_i9_10900K_AES_Key192;
  P_Intel_Core_i9_10900K_AES_Key256 =^T_Intel_Core_i9_10900K_AES_Key256;
  P_Intel_Core_i9_10900K_AES_ExpandedKey128
=^T_Intel_Core_i9_10900K_AES_ExpandedKey128;
  P_Intel_Core_i9_10900K_AES_ExpandedKey192
=^T_Intel_Core_i9_10900K_AES_ExpandedKey192;
  P_Intel_Core_i9_10900K_AES_ExpandedKey256
=^T_Intel_Core_i9_10900K_AES_ExpandedKey256;

// Розширення ключа для шифрування

procedure Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(const Key:
T_Intel_Core_i9_10900K_AES_Key128;
  var ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128); overload;
procedure Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(const Key:
T_Intel_Core_i9_10900K_AES_Key192;
  var ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192); overload;
procedure Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(const Key:
T_Intel_Core_i9_10900K_AES_Key256;
  var ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256); overload;

// Блок раундів шифрування

procedure Encrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey128;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer); overload;
procedure Encrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey192;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer); overload;
procedure Encrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey256;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer); overload;

// Поток раундів Шифрування (ECB режим)

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; Dest: TStream); overload;
```

```
procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128; Dest: TStream);
overload;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; Dest: TStream); overload;
procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192; Dest: TStream);
overload;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key256; Dest: TStream); overload;
procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256; Dest: TStream);
overload;

// Поток раундів шифрування (CBC режим)

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream); overload;
procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream); overload;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream); overload;
procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream); overload;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key256; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream); overload;
procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream); overload;

// Перетворення сеансового ключа для дешифрування

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey128); overload;
procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(const Key:
T_Intel_Core_i9_10900K_AES_Key128;
  var ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128); overload;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey192); overload;
procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(const Key:
T_Intel_Core_i9_10900K_AES_Key192;
  var ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192); overload;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey256); overload;
```

```
procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(const Key:
T_Intel_Core_i9_10900K_AES_Key256;
  var ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256); overload;

// Блок раундів дешифрування

procedure Decrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey128;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey192;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey256;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer); overload;

// Поток раундів дешифрування (ECB режим)

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; Dest: TStream); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128; Dest: TStream);
overload;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; Dest: TStream); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192; Dest: TStream);
overload;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key256; Dest: TStream); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256; Dest: TStream);
overload;

// Поток раундів дешифрування (CBC режим)

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream); overload;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream); overload;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
```

```
  const Key: T_Intel_Core_i9_10900K_AES_Key256; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream); overload;
procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream); overload;

resourcestring
  SInvalidInBufSize = 'Не хватає розміру буферу длядешифрування';
  SReadError = 'Помилка читання потоку';
  SWriteError = 'Помилка запису потоку';

implementation

type
  PLongWord = ^LongWord;

function Min(A, B: integer): integer;
begin
  if A < B then
    Result := A
  else
    Result := B;
end;

const
  Rcon: array [1..30] of longword = (
    $00000001, $00000002, $00000004, $00000008, $00000010, $00000020,
    $00000040, $00000080, $0000001B, $00000036, $0000006C, $000000D8,
    $000000AB, $0000004D, $0000009A, $0000002F, $0000005E, $000000BC,
    $00000063, $000000C6, $00000097, $00000035, $0000006A, $000000D4,
    $000000B3, $0000007D, $000000FA, $000000EF, $000000C5, $00000091
  );

//Таблиці перестановки

  ForwardTable: array [0..255] of longword = (
    $A56363C6, $847C7CF8, $997777EE, $8D7B7BF6, $0DF2F2FF, $BD6B6BD6, $B16F6FDE,
$54C5C591,
    $50303060, $03010102, $A96767CE, $7D2B2B56, $19FEFEE7, $62D7D7B5, $E6ABAB4D,
$9A7676EC,
    $45CACA8F, $9D82821F, $40C9C989, $877D7DFA, $15FAFAEF, $EB5959B2, $C947478E,
$0BF0F0FB,
    $ECADAD41, $67D4D4B3, $FDA2A25F, $EAAFAF45, $BF9C9C23, $F7A4A453, $967272E4,
$5BC0C09B,
    $C2B7B775, $1CFDFDE1, $AE93933D, $6A26264C, $5A36366C, $413F3F7E, $02F7F7F5,
$4FCCCC83,
    $5C343468, $F4A5A551, $34E5E5D1, $08F1F1F9, $937171E2, $73D8D8AB, $53313162,
$3F15152A,
    $0C040408, $52C7C795, $65232346, $5EC3C39D, $28181830, $A1969637, $0F05050A,
$B59A9A2F,
    $0907070E, $36121224, $9B80801B, $3DE2E2DF, $26EBEBCD, $6927274E, $CDB2B27F,
$9F7575EA,
    $1B090912, $9E83831D, $742C2C58, $2E1A1A34, $2D1B1B36, $B26E6EDC, $EE5A5AB4,
$FBA0A05B,
    $F65252A4, $4D3B3B76, $61D6D6B7, $CEB3B37D, $7B292952, $3EE3E3DD, $712F2F5E,
$97848413,
    $F55353A6, $68D1D1B9, $00000000, $2CEDEDC1, $60202040, $1FFCFCE3, $C8B1B179,
$ED5B5BB6,
    $BE6A6AD4, $46CBCB8D, $D9BEBE67, $4B393972, $DE4A4A94, $D44C4C98, $E85858B0,
$4ACFCF85,
    $6BD0D0BB, $2AEFEFC5, $E5AAAA4F, $16FBFBED, $C5434386, $D74D4D9A, $55333366,
$94858511,
    $CF45458A, $10F9F9E9, $06020204, $817F7FFE, $F05050A0, $443C3C78, $BA9F9F25,
$E3A8A84B,
    $F35151A2, $FEA3A35D, $C0404080, $8A8F8F05, $AD92923F, $BC9D9D21, $48383870,
$04F5F5F1,
```

```
    $DFBCBC63, $C1B6B677, $75DADAAF, $63212142, $30101020, $1AFFFFE5, $0EF3F3FD,
$6DD2D2BF,
    $4CCDCD81, $140C0C18, $35131326, $2FECECC3, $E15F5FBE, $A2979735, $CC444488,
$3917172E,
    $57C4C493, $F2A7A755, $827E7EFC, $473D3D7A, $AC6464C8, $E75D5DBA, $2B191932,
$957373E6,
    $A06060C0, $98818119, $D14F4F9E, $7FDCDCA3, $66222244, $7E2A2A54, $AB90903B,
$8388880B,
    $CA46468C, $29EEEEC7, $D3B8B86B, $3C141428, $79DEDEA7, $E25E5EBC, $1D0B0B16,
$76DBDBAD,
    $3BE0E0DB, $56323264, $4E3A3A74, $1E0A0A14, $DB494992, $0A06060C, $6C242448,
$E45C5CB8,
    $5DC2C29F, $6ED3D3BD, $EFACAC43, $A66262C4, $A8919139, $A4959531, $37E4E4D3,
$8B7979F2,
    $32E7E7D5, $43C8C88B, $5937376E, $B76D6DDA, $8C8D8D01, $64D5D5B1, $D24E4E9C,
$E0A9A949,
    $B46C6CD8, $FA5656AC, $07F4F4F3, $25EAEACF, $AF6565CA, $8E7A7AF4, $E9AEAE47,
$18080810,
    $D5BABA6F, $887878F0, $6F25254A, $722E2E5C, $241C1C38, $F1A6A657, $C7B4B473,
$51C6C697,
    $23E8E8CB, $7CDDDDA1, $9C7474E8, $211F1F3E, $DD4B4B96, $DCBDBD61, $868B8B0D,
$858A8A0F,
    $907070E0, $423E3E7C, $C4B5B571, $AA6666CC, $D8484890, $05030306, $01F6F6F7,
$120E0E1C,
    $A36161C2, $5F35356A, $F95757AE, $D0B9B969, $91868617, $58C1C199, $271D1D3A,
$B99E9E27,
    $38E1E1D9, $13F8F8EB, $B398982B, $33111122, $BB6969D2, $70D9D9A9, $898E8E07,
$A7949433,
    $B69B9B2D, $221E1E3C, $92878715, $20E9E9C9, $49CECE87, $FF5555AA, $78282850,
$7ADFDFA5,
    $8F8C8C03, $F8A1A159, $80898909, $170D0D1A, $DABFBF65, $31E6E6D7, $C6424284,
$B86868D0,
    $C3414182, $B0999929, $772D2D5A, $110F0F1E, $CBB0B07B, $FC5454A8, $D6BBBB6D,
$3A16162C
  );

  LastForwardTable: array [0..255] of longword = (
    $00000063, $0000007C, $00000077, $0000007B, $000000F2, $0000006B, $0000006F,
$000000C5,
    $00000030, $00000001, $00000067, $0000002B, $000000FE, $000000D7, $000000AB,
$00000076,
    $000000CA, $00000082, $000000C9, $0000007D, $000000FA, $00000059, $00000047,
$000000F0,
    $000000AD, $000000D4, $000000A2, $000000AF, $0000009C, $000000A4, $00000072,
$000000C0,
    $000000B7, $000000FD, $00000093, $00000026, $00000036, $0000003F, $000000F7,
$000000CC,
    $00000034, $000000A5, $000000E5, $000000F1, $00000071, $000000D8, $00000031,
$00000015,
    $00000004, $000000C7, $00000023, $000000C3, $00000018, $00000096, $00000005,
$0000009A,
    $00000007, $00000012, $00000080, $000000E2, $000000EB, $00000027, $000000B2,
$00000075,
    $00000009, $00000083, $0000002C, $0000001A, $0000001B, $0000006E, $0000005A,
$000000A0,
    $00000052, $0000003B, $000000D6, $000000B3, $00000029, $000000E3, $0000002F,
$00000084,
    $00000053, $000000D1, $00000000, $000000ED, $00000020, $000000FC, $000000B1,
$0000005B,
    $0000006A, $000000CB, $000000BE, $00000039, $0000004A, $0000004C, $00000058,
$000000CF,
    $000000D0, $000000EF, $000000AA, $000000FB, $00000043, $0000004D, $00000033,
$00000085,
    $00000045, $000000F9, $00000002, $0000007F, $00000050, $0000003C, $0000009F,
$000000A8,
    $00000051, $000000A3, $00000040, $0000008F, $00000092, $0000009D, $00000038,
$000000F5,
    $000000BC, $000000B6, $000000DA, $00000021, $00000010, $000000FF, $000000F3,
$000000D2,
```

```
    $000000CD, $0000000C, $00000013, $000000EC, $0000005F, $00000097, $00000044,
$00000017,
    $000000C4, $000000A7, $0000007E, $0000003D, $00000064, $0000005D, $00000019,
$00000073,
    $00000060, $00000081, $0000004F, $000000DC, $00000022, $0000002A, $00000090,
$00000088,
    $00000046, $000000EE, $000000B8, $00000014, $000000DE, $0000005E, $0000000B,
$000000DB,
    $000000E0, $00000032, $0000003A, $0000000A, $00000049, $00000006, $00000024,
$0000005C,
    $000000C2, $000000D3, $000000AC, $00000062, $00000091, $00000095, $000000E4,
$00000079,
    $000000E7, $000000C8, $00000037, $0000006D, $0000008D, $000000D5, $0000004E,
$000000A9,
    $0000006C, $00000056, $000000F4, $000000EA, $00000065, $0000007A, $000000AE,
$00000008,
    $000000BA, $00000078, $00000025, $0000002E, $0000001C, $000000A6, $000000B4,
$000000C6,
    $000000E8, $000000DD, $00000074, $0000001F, $0000004B, $000000BD, $0000008B,
$0000008A,
    $00000070, $0000003E, $000000B5, $00000066, $00000048, $00000003, $000000F6,
$0000000E,
    $00000061, $00000035, $00000057, $000000B9, $00000086, $000000C1, $0000001D,
$0000009E,
    $000000E1, $000000F8, $00000098, $00000011, $00000069, $000000D9, $0000008E,
$00000094,
    $0000009B, $0000001E, $00000087, $000000E9, $000000CE, $00000055, $00000028,
$000000DF,
    $0000008C, $000000A1, $00000089, $0000000D, $000000BF, $000000E6, $00000042,
$00000068,
    $00000041, $00000099, $0000002D, $0000000F, $000000B0, $00000054, $000000BB,
$00000016
  );

  InverseTable: array [0..255] of longword = (
    $50A7F451, $5365417E, $C3A4171A, $965E273A, $CB6BAB3B, $F1459D1F, $AB58FAAC,
$9303E34B,
    $55FA3020, $F66D76AD, $9176CC88, $254C02F5, $FCD7E54F, $D7CB2AC5, $80443526,
$8FA362B5,
    $495AB1DE, $671BBA25, $980EEA45, $E1C0FE5D, $02752FC3, $12F04C81, $A397468D,
$C6F9D36B,
    $E75F8F03, $959C9215, $EB7A6DBF, $DA595295, $2D83BED4, $D3217458, $2969E049,
$44C8C98E,
    $6A89C275, $78798EF4, $6B3E5899, $DD71B927, $B64FE1BE, $17AD88F0, $66AC20C9,
$B43ACE7D,
    $184ADF63, $82311AE5, $60335197, $457F5362, $E07764B1, $84AE6BBB, $1CA081FE,
$942B08F9,
    $58684870, $19FD458F, $876CDE94, $B7F87B52, $23D373AB, $E2024B72, $578F1FE3,
$2AAB5566,
    $0728EBB2, $03C2B52F, $9A7BC586, $A50837D3, $F2872830, $B2A5BF23, $BA6A0302,
$5C8216ED,
    $2B1CCF8A, $92B479A7, $F0F207F3, $A1E2694E, $CDF4DA65, $D5BE0506, $1F6234D1,
$8AFEA6C4,
    $9D532E34, $A055F3A2, $32E18A05, $75EBF6A4, $39EC830B, $AAEF6040, $069F715E,
$51106EBD,
    $F98A213E, $3D06DD96, $AE053EDD, $46BDE64D, $B58D5491, $055DC471, $6FD40604,
$FF155060,
    $24FB9819, $97E9BDD6, $CC434089, $779ED967, $BD42E8B0, $888B8907, $385B19E7,
$DBEEC879,
    $470A7CA1, $E90F427C, $C91E84F8, $00000000, $83868009, $48ED2B32, $AC70111E,
$4E725A6C,
    $FBFF0EFD, $5638850F, $1ED5AE3D, $27392D36, $64D90F0A, $21A65C68, $D1545B9B,
$3A2E3624,
    $B1670A0C, $0FE75793, $D296EEB4, $9E919B1B, $4FC5C080, $A220DC61, $694B775A,
$161A121C,
    $0ABA93E2, $E52AA0C0, $43E0223C, $1D171B12, $0B0D090E, $ADC78BF2, $B9A8B62D,
$C8A91E14,
    $8519F157, $4C0775AF, $BBDD99EE, $FD607FA3, $9F2601F7, $BCF5725C, $C53B6644,
$347EFB5B,
```

```
    $7629438B, $DCC623CB, $68FCEDB6, $63F1E4B8, $CADC31D7, $10856342, $40229713,
$2011C684,
    $7D244A85, $F83DBBD2, $1132F9AE, $6DA129C7, $4B2F9E1D, $F330B2DC, $EC52860D,
$D0E3C177,
    $6C16B32B, $99B970A9, $FA489411, $2264E947, $C48CFCA8, $1A3FF0A0, $D82C7D56,
$EF903322,
    $C74E4987, $C1D138D9, $FEA2CA8C, $360BD498, $CF81F5A6, $28DE7AA5, $268EB7DA,
$A4BFAD3F,
    $E49D3A2C, $0D927850, $9BCC5F6A, $62467E54, $C2138DF6, $E8B8D890, $5EF7392E,
$F5AFC382,
    $BE805D9F, $7C93D069, $A92DD56F, $B31225CF, $3B99ACC8, $A77D1810, $6E639CE8,
$7BBB3BDB,
    $097826CD, $F418596E, $01B79AEC, $A89A4F83, $656E95E6, $7EE6FFAA, $08CFBC21,
$E6E815EF,
    $D99BE7BA, $CE366F4A, $D4099FEA, $D67CB029, $AFB2A431, $31233F2A, $3094A5C6,
$C066A235,
    $37BC4E74, $A6CA82FC, $B0D090E0, $15D8A733, $4A9804F1, $F7DAEC41, $0E50CD7F,
$2FF69117,
    $8DD64D76, $4DB0EF43, $544DAACC, $DF0496E4, $E3B5D19E, $1B886A4C, $B81F2CC1,
$7F516546,
    $04EA5E9D, $5D358C01, $737487FA, $2E410BFB, $5A1D67B3, $52D2DB92, $335610E9,
$1347D66D,
    $8C61D79A, $7A0CA137, $8E14F859, $893C13EB, $EE27A9CE, $35C961B7, $EDE51CE1,
$3CB1477A,
    $59DFD29C, $3F73F255, $79CE1418, $BF37C773, $EACDF753, $5BAAFD5F, $146F3DDF,
$86DB4478,
    $81F3AFCA, $3EC468B9, $2C342438, $5F40A3C2, $72C31D16, $0C25E2BC, $8B493C28,
$41950DFF,
    $7101A839, $DEB30C08, $9CE4B4D8, $90C15664, $6184CB7B, $70B632D5, $745C6C48,
$4257B8D0
  );

  LastInverseTable: array [0..255] of longword = (
    $00000052, $00000009, $0000006A, $000000D5, $00000030, $00000036, $000000A5,
$00000038,
    $000000BF, $00000040, $000000A3, $0000009E, $00000081, $000000F3, $000000D7,
$000000FB,
    $0000007C, $000000E3, $00000039, $00000082, $0000009B, $0000002F, $000000FF,
$00000087,
    $00000034, $0000008E, $00000043, $00000044, $000000C4, $000000DE, $000000E9,
$000000CB,
    $00000054, $0000007B, $00000094, $00000032, $000000A6, $000000C2, $00000023,
$0000003D,
    $000000EE, $0000004C, $00000095, $0000000B, $00000042, $000000FA, $000000C3,
$0000004E,
    $00000008, $0000002E, $000000A1, $00000066, $00000028, $000000D9, $00000024,
$000000B2,
    $00000076, $0000005B, $000000A2, $00000049, $0000006D, $0000008B, $000000D1,
$00000025,
    $00000072, $000000F8, $000000F6, $00000064, $00000086, $00000068, $00000098,
$00000016,
    $000000D4, $000000A4, $0000005C, $000000CC, $0000005D, $00000065, $000000B6,
$00000092,
    $0000006C, $00000070, $00000048, $00000050, $000000FD, $000000ED, $000000B9,
$000000DA,
    $0000005E, $00000015, $00000046, $00000057, $000000A7, $0000008D, $0000009D,
$00000084,
    $00000090, $000000D8, $000000AB, $00000000, $0000008C, $000000BC, $000000D3,
$0000000A,
    $000000F7, $000000E4, $00000058, $00000005, $000000B8, $000000B3, $00000045,
$00000006,
    $000000D0, $0000002C, $0000001E, $0000008F, $000000CA, $0000003F, $0000000F,
$00000002,
    $000000C1, $000000AF, $000000BD, $00000003, $00000001, $00000013, $0000008A,
$0000006B,
    $0000003A, $00000091, $00000011, $00000041, $0000004F, $00000067, $000000DC,
$000000EA,
    $00000097, $000000F2, $000000CF, $000000CE, $000000F0, $000000B4, $000000E6,
$00000073,
```

```
    $00000096, $000000AC, $00000074, $00000022, $000000E7, $000000AD, $00000035,
$00000085,
    $000000E2, $000000F9, $00000037, $000000E8, $0000001C, $00000075, $000000DF,
$0000006E,
    $00000047, $000000F1, $0000001A, $00000071, $0000001D, $00000029, $000000C5,
$00000089,
    $0000006F, $000000B7, $00000062, $0000000E, $000000AA, $00000018, $000000BE,
$0000001B,
    $000000FC, $00000056, $0000003E, $0000004B, $000000C6, $000000D2, $00000079,
$00000020,
    $0000009A, $000000DB, $000000C0, $000000FE, $00000078, $000000CD, $0000005A,
$000000F4,
    $0000001F, $000000DD, $000000A8, $00000033, $00000088, $00000007, $000000C7,
$00000031,
    $000000B1, $00000012, $00000010, $00000059, $00000027, $00000080, $000000EC,
$0000005F,
    $00000060, $00000051, $0000007F, $000000A9, $00000019, $000000B5, $0000004A,
$0000000D,
    $0000002D, $000000E5, $0000007A, $0000009F, $00000093, $000000C9, $0000009C,
$000000EF,
    $000000A0, $000000E0, $0000003B, $0000004D, $000000AE, $0000002A, $000000F5,
$000000B0,
    $000000C8, $000000EB, $000000BB, $0000003C, $00000083, $00000053, $00000099,
$00000061,
    $00000017, $0000002B, $00000004, $0000007E, $000000BA, $00000077, $000000D6,
$00000026,
    $000000E1, $00000069, $00000014, $00000063, $00000055, $00000021, $0000000C,
$0000007D
  );

//Розширення ключа для шифрування

procedure Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(const Key:
T_Intel_Core_i9_10900K_AES_Key128; var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey128);
var
  I, J: integer;
  T: longword;
  W0, W1, W2, W3: longword;
begin
  ExpandedKey[0] := PLongWord(@Key[0])^;
  ExpandedKey[1] := PLongWord(@Key[4])^;
  ExpandedKey[2] := PLongWord(@Key[8])^;
  ExpandedKey[3] := PLongWord(@Key[12])^;
  I := 0; J := 1;
  repeat
    T := (ExpandedKey[I + 3] shl 24) or (ExpandedKey[I + 3] shr 8);
    W0 := LastForwardTable[Byte(T)]; W1 := LastForwardTable[Byte(T shr 8)];
    W2 := LastForwardTable[Byte(T shr 16)]; W3 := LastForwardTable[Byte(T shr
24)];
    ExpandedKey[I + 4] := ExpandedKey[I] xor
      (W0 xor ((W1 shl 8) or (W1 shr 24)) xor
      ((W2 shl 16) or (W2 shr 16)) xor ((W3 shl 24) or (W3 shr 8))) xor Rcon[J];
    Inc(J);
    ExpandedKey[I + 5] := ExpandedKey[I + 1] xor ExpandedKey[I + 4];
    ExpandedKey[I + 6] := ExpandedKey[I + 2] xor ExpandedKey[I + 5];
    ExpandedKey[I + 7] := ExpandedKey[I + 3] xor ExpandedKey[I + 6];
    Inc(I, 4);
  until I >= 40;
end;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(const Key:
T_Intel_Core_i9_10900K_AES_Key192; var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey192); overload;
var
  I, J: integer;
  T: longword;
  W0, W1, W2, W3: longword;
begin
```

```
  ExpandedKey[0] := PLongWord(@Key[0])^;
  ExpandedKey[1] := PLongWord(@Key[4])^;
  ExpandedKey[2] := PLongWord(@Key[8])^;
  ExpandedKey[3] := PLongWord(@Key[12])^;
  ExpandedKey[4] := PLongWord(@Key[16])^;
  ExpandedKey[5] := PLongWord(@Key[20])^;
  I := 0; J := 1;
  repeat
    T := (ExpandedKey[I + 5] shl 24) or (ExpandedKey[I + 5] shr 8);
    W0 := LastForwardTable[Byte(T)]; W1 := LastForwardTable[Byte(T shr 8)];
    W2 := LastForwardTable[Byte(T shr 16)]; W3 := LastForwardTable[Byte(T shr
24)];
    ExpandedKey[I + 6] := ExpandedKey[I] xor
      (W0 xor ((W1 shl 8) or (W1 shr 24)) xor
      ((W2 shl 16) or (W2 shr 16)) xor ((W3 shl 24) or (W3 shr 8))) xor Rcon[J];
    Inc(J);
    ExpandedKey[I + 7] := ExpandedKey[I + 1] xor ExpandedKey[I + 6];
    ExpandedKey[I + 8] := ExpandedKey[I + 2] xor ExpandedKey[I + 7];
    ExpandedKey[I + 9] := ExpandedKey[I + 3] xor ExpandedKey[I + 8];
    ExpandedKey[I + 10] := ExpandedKey[I + 4] xor ExpandedKey[I + 9];
    ExpandedKey[I + 11] := ExpandedKey[I + 5] xor ExpandedKey[I + 10];
    Inc(I, 6);
  until I >= 46;
end;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(const Key:
T_Intel_Core_i9_10900K_AES_Key256; var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey256); overload;
var
  I, J: integer;
  T: longword;
  W0, W1, W2, W3: longword;
begin
  ExpandedKey[0] := PLongWord(@Key[0])^;
  ExpandedKey[1] := PLongWord(@Key[4])^;
  ExpandedKey[2] := PLongWord(@Key[8])^;
  ExpandedKey[3] := PLongWord(@Key[12])^;
  ExpandedKey[4] := PLongWord(@Key[16])^;
  ExpandedKey[5] := PLongWord(@Key[20])^;
  ExpandedKey[6] := PLongWord(@Key[24])^;
  ExpandedKey[7] := PLongWord(@Key[28])^;
  I := 0; J := 1;
  repeat
    T := (ExpandedKey[I + 7] shl 24) or (ExpandedKey[I + 7] shr 8);
    W0 := LastForwardTable[Byte(T)]; W1 := LastForwardTable[Byte(T shr 8)];
    W2 := LastForwardTable[Byte(T shr 16)]; W3 := LastForwardTable[Byte(T shr
24)];
    ExpandedKey[I + 8] := ExpandedKey[I] xor
      (W0 xor ((W1 shl 8) or (W1 shr 24)) xor
      ((W2 shl 16) or (W2 shr 16)) xor ((W3 shl 24) or (W3 shr 8))) xor Rcon[J];
    Inc(J);
    ExpandedKey[I + 9] := ExpandedKey[I + 1] xor ExpandedKey[I + 8];
    ExpandedKey[I + 10] := ExpandedKey[I + 2] xor ExpandedKey[I + 9];
    ExpandedKey[I + 11] := ExpandedKey[I + 3] xor ExpandedKey[I + 10];
    W0 := LastForwardTable[Byte(ExpandedKey[I + 11])];
    W1 := LastForwardTable[Byte(ExpandedKey[I + 11] shr 8)];
    W2 := LastForwardTable[Byte(ExpandedKey[I + 11] shr 16)];
    W3 := LastForwardTable[Byte(ExpandedKey[I + 11] shr 24)];
    ExpandedKey[I + 12] := ExpandedKey[I + 4] xor
      (W0 xor ((W1 shl 8) or (W1 shr 24)) xor
      ((W2 shl 16) or (W2 shr 16)) xor ((W3 shl 24) or (W3 shr 8)));
    ExpandedKey[I + 13] := ExpandedKey[I + 5] xor ExpandedKey[I + 12];
    ExpandedKey[I + 14] := ExpandedKey[I + 6] xor ExpandedKey[I + 13];
    ExpandedKey[I + 15] := ExpandedKey[I + 7] xor ExpandedKey[I + 14];
    Inc(I, 8);
  until I >= 52;
end;
```

```
//Процедура шифрування

procedure Encrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey128;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer);
var
  T0, T1: array [0..3] of longword;
  W0, W1, W2, W3: longword;
begin
  // Ініціалізація
  T0[0] := PLongWord(@InBuf[0])^ xor Key[0];
  T0[1] := PLongWord(@InBuf[4])^ xor Key[1];
  T0[2] := PLongWord(@InBuf[8])^ xor Key[2];
  T0[3] := PLongWord(@InBuf[12])^ xor Key[3];
  // Попередня трансформація - 9 раз
  // раунд 1
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[4];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[5];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[6];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[7];
  // раунд 2
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[8];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[9];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[10];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[11];
  // раунд 3
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[12];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
```

```
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[13];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[14];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[15];
  // раунд 4
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[16];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[17];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[18];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[19];
  // раунд 5
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[20];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[21];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[22];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[23];
  // раунд 6
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[24];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[25];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[26];
```

```
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[27];
  // раунд 7
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[28];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[29];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[30];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[31];
  // раунд 8
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[32];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[33];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[34];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[35];
  // раунд 9
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[36];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[37];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[38];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[39];
  // останій раунд перетворень
```

```
  W0 := LastForwardTable[Byte(T1[0])]; W1 := LastForwardTable[Byte(T1[1] shr
8)];
  W2 := LastForwardTable[Byte(T1[2] shr 16)]; W3 := LastForwardTable[Byte(T1[3]
shr 24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[40];
  W0 := LastForwardTable[Byte(T1[1])]; W1 := LastForwardTable[Byte(T1[2] shr
8)];
  W2 := LastForwardTable[Byte(T1[3] shr 16)]; W3 := LastForwardTable[Byte(T1[0]
shr 24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[41];
  W0 := LastForwardTable[Byte(T1[2])]; W1 := LastForwardTable[Byte(T1[3] shr
8)];
  W2 := LastForwardTable[Byte(T1[0] shr 16)]; W3 := LastForwardTable[Byte(T1[1]
shr 24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[42];
  W0 := LastForwardTable[Byte(T1[3])]; W1 := LastForwardTable[Byte(T1[0] shr
8)];
  W2 := LastForwardTable[Byte(T1[1] shr 16)]; W3 := LastForwardTable[Byte(T1[2]
shr 24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[43];
  // кінець роботи алгоритму
  PLongWord(@OutBuf[0])^ := T0[0]; PLongWord(@OutBuf[4])^ := T0[1];
  PLongWord(@OutBuf[8])^ := T0[2]; PLongWord(@OutBuf[12])^ := T0[3];
end;


procedure Encrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey192;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer);
var
  T0, T1: array [0..3] of longword;
  W0, W1, W2, W3: longword;
begin
  // ініціалізація
  T0[0] := PLongWord(@InBuf[0])^ xor Key[0];
  T0[1] := PLongWord(@InBuf[4])^ xor Key[1];
  T0[2] := PLongWord(@InBuf[8])^ xor Key[2];
  T0[3] := PLongWord(@InBuf[12])^ xor Key[3];
  // Попередня трансформація - 11 раз
  // раунд 1
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[4];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[5];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[6];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[7];
  // раунд 2
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
```

```
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[8];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[9];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[10];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[11];
  // раунд 3
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[12];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[13];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[14];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[15];
  // раунд 4
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[16];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[17];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[18];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[19];
  // раунд 5
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[20];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
```

```
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[21];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[22];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[23];
  // раунд 6
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[24];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[25];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[26];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[27];
  // раунд 7
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[28];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[29];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[30];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[31];
  // раунд 8
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[32];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[33];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[34];
```

```
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[35];
  // раунд 9
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[36];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[37];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[38];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[39];
  // раунд 10
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[40];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[41];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[42];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[43];
  // раунд 11
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[44];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[45];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[46];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[47];
  // останій раунд перетворень
```

```
  W0 := LastForwardTable[Byte(T1[0])]; W1 := LastForwardTable[Byte(T1[1] shr
8)];
  W2 := LastForwardTable[Byte(T1[2] shr 16)]; W3 := LastForwardTable[Byte(T1[3]
shr 24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[48];
  W0 := LastForwardTable[Byte(T1[1])]; W1 := LastForwardTable[Byte(T1[2] shr
8)];
  W2 := LastForwardTable[Byte(T1[3] shr 16)]; W3 := LastForwardTable[Byte(T1[0]
shr 24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[49];
  W0 := LastForwardTable[Byte(T1[2])]; W1 := LastForwardTable[Byte(T1[3] shr
8)];
  W2 := LastForwardTable[Byte(T1[0] shr 16)]; W3 := LastForwardTable[Byte(T1[1]
shr 24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[50];
  W0 := LastForwardTable[Byte(T1[3])]; W1 := LastForwardTable[Byte(T1[0] shr
8)];
  W2 := LastForwardTable[Byte(T1[1] shr 16)]; W3 := LastForwardTable[Byte(T1[2]
shr 24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[51];
  // кінець роботи алгоритму
  PLongWord(@OutBuf[0])^ := T0[0]; PLongWord(@OutBuf[4])^ := T0[1];
  PLongWord(@OutBuf[8])^ := T0[2]; PLongWord(@OutBuf[12])^ := T0[3];
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey256;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer);
var
  T0, T1: array [0..3] of longword;
  W0, W1, W2, W3: longword;
begin
  // ініціалізація
  T0[0] := PLongWord(@InBuf[0])^ xor Key[0];
  T0[1] := PLongWord(@InBuf[4])^ xor Key[1];
  T0[2] := PLongWord(@InBuf[8])^ xor Key[2];
  T0[3] := PLongWord(@InBuf[12])^ xor Key[3];
  // Попередня трансформація 13 разів
  // раунд 1
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[4];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[5];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[6];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[7];
  // раунд 2
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
```

```
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[8];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[9];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[10];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[11];
  // раунд 3
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[12];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[13];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[14];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[15];
  // раунд 4
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[16];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[17];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[18];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[19];
  // раунд 5
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[20];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[21];
```

```
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[22];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[23];
  // раунд 6
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[24];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[25];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[26];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[27];
  // раунд 7
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[28];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[29];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[30];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[31];
  // раунд 8
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[32];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[33];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[34];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
```

```
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[35];
  // раунд 9
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[36];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[37];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[38];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[39];
  // раунд 10
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[40];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[41];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[42];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[43];
  // раунд 11
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[44];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[45];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[46];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[47];
  // раунд 12
  W0 := ForwardTable[Byte(T1[0])]; W1 := ForwardTable[Byte(T1[1] shr 8)];
```

```
  W2 := ForwardTable[Byte(T1[2] shr 16)]; W3 := ForwardTable[Byte(T1[3] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[48];
  W0 := ForwardTable[Byte(T1[1])]; W1 := ForwardTable[Byte(T1[2] shr 8)];
  W2 := ForwardTable[Byte(T1[3] shr 16)]; W3 := ForwardTable[Byte(T1[0] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[49];
  W0 := ForwardTable[Byte(T1[2])]; W1 := ForwardTable[Byte(T1[3] shr 8)];
  W2 := ForwardTable[Byte(T1[0] shr 16)]; W3 := ForwardTable[Byte(T1[1] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[50];
  W0 := ForwardTable[Byte(T1[3])]; W1 := ForwardTable[Byte(T1[0] shr 8)];
  W2 := ForwardTable[Byte(T1[1] shr 16)]; W3 := ForwardTable[Byte(T1[2] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[51];
  // раунд 13
  W0 := ForwardTable[Byte(T0[0])]; W1 := ForwardTable[Byte(T0[1] shr 8)];
  W2 := ForwardTable[Byte(T0[2] shr 16)]; W3 := ForwardTable[Byte(T0[3] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[52];
  W0 := ForwardTable[Byte(T0[1])]; W1 := ForwardTable[Byte(T0[2] shr 8)];
  W2 := ForwardTable[Byte(T0[3] shr 16)]; W3 := ForwardTable[Byte(T0[0] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[53];
  W0 := ForwardTable[Byte(T0[2])]; W1 := ForwardTable[Byte(T0[3] shr 8)];
  W2 := ForwardTable[Byte(T0[0] shr 16)]; W3 := ForwardTable[Byte(T0[1] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[54];
  W0 := ForwardTable[Byte(T0[3])]; W1 := ForwardTable[Byte(T0[0] shr 8)];
  W2 := ForwardTable[Byte(T0[1] shr 16)]; W3 := ForwardTable[Byte(T0[2] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[55];
  // останій раунд перетворень
  W0 := LastForwardTable[Byte(T1[0])]; W1 := LastForwardTable[Byte(T1[1] shr
8)];
  W2 := LastForwardTable[Byte(T1[2] shr 16)]; W3 := LastForwardTable[Byte(T1[3]
shr 24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[56];
  W0 := LastForwardTable[Byte(T1[1])]; W1 := LastForwardTable[Byte(T1[2] shr
8)];
  W2 := LastForwardTable[Byte(T1[3] shr 16)]; W3 := LastForwardTable[Byte(T1[0]
shr 24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[57];
  W0 := LastForwardTable[Byte(T1[2])]; W1 := LastForwardTable[Byte(T1[3] shr
8)];
  W2 := LastForwardTable[Byte(T1[0] shr 16)]; W3 := LastForwardTable[Byte(T1[1]
shr 24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[58];
  W0 := LastForwardTable[Byte(T1[3])]; W1 := LastForwardTable[Byte(T1[0] shr
8)];
  W2 := LastForwardTable[Byte(T1[1] shr 16)]; W3 := LastForwardTable[Byte(T1[2]
shr 24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[59];
  // кінець роботи алгоритму
  PLongWord(@OutBuf[0])^ := T0[0]; PLongWord(@OutBuf[4])^ := T0[1];
  PLongWord(@OutBuf[8])^ := T0[2]; PLongWord(@OutBuf[12])^ := T0[3];
end;
```

```
//Розширення ключа для дешифрування

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey128);
var
  I: integer;
  U, F2, F4, F8, F9: longword;
begin
  for I := 1 to 9 do
  begin
    F9 := ExpandedKey[I * 4];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 1];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 1] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 2];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 2] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 3];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 3] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
  end;
end;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(const Key:
T_Intel_Core_i9_10900K_AES_Key128; var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey128);
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(ExpandedKey);
end;
```

```
procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey192);
var
  I: integer;
  U, F2, F4, F8, F9: longword;
begin
  for I := 1 to 11 do
  begin
    F9 := ExpandedKey[I * 4];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 1];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 1] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 2];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 2] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 3];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 3] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
  end;
end;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(const Key:
T_Intel_Core_i9_10900K_AES_Key192; var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey192);
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(ExpandedKey);
end;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey256);
```

```
var
  I: integer;
  U, F2, F4, F8, F9: longword;
begin
  for I := 1 to 13 do
  begin
    F9 := ExpandedKey[I * 4];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 1];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 1] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 2];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 2] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
    F9 := ExpandedKey[I * 4 + 3];
    U := F9 and $80808080;
    F2 := ((F9 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F2 and $80808080;
    F4 := ((F2 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    U := F4 and $80808080;
    F8 := ((F4 and $7F7F7F7F) shl 1) xor ((U - (U shr 7)) and $1B1B1B1B);
    F9 := F9 xor F8;
    ExpandedKey[I * 4 + 3] := F2 xor F4 xor F8 xor
      (((F2 xor F9) shl 24) or ((F2 xor F9) shr 8)) xor
      (((F4 xor F9) shl 16) or ((F4 xor F9) shr 16)) xor ((F9 shl 8) or (F9 shr
24));
  end;
end;

procedure Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(const Key:
T_Intel_Core_i9_10900K_AES_Key256; var ExpandedKey:
T_Intel_Core_i9_10900K_AES_ExpandedKey256);
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(ExpandedKey);
end;

//Процедура дешифрування
```

```
procedure Decrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey128;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer);
var
  T0, T1: array [0..3] of longword;
  W0, W1, W2, W3: longword;
begin
  // ініціалізація
  T0[0] := PLongWord(@InBuf[0])^ xor Key[40];
  T0[1] := PLongWord(@InBuf[4])^ xor Key[41];
  T0[2] := PLongWord(@InBuf[8])^ xor Key[42];
  T0[3] := PLongWord(@InBuf[12])^ xor Key[43];
  // Попередня трансформація  9 разів
  // раунд 1
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[36];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[37];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[38];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[39];
  // раунд 2
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[32];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[33];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[34];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[35];
  // раунд 3
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[28];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[29];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
```

```
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[30];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[31];
  // раунд 4
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[24];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[25];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[26];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[27];
  // раунд 5
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[20];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[21];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[22];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[23];
  // раунд 6
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[16];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[17];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[18];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
```

```
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[19];
  // раунд 7
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[12];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[13];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[14];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[15];
  // раунд 8
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[8];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[9];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[10];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[11];
  // раунд 9
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[4];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[5];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[6];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[7];
  // останій раунд перетворень
```

```
  W0 := LastInverseTable[Byte(T1[0])]; W1 := LastInverseTable[Byte(T1[3] shr
8)];
  W2 := LastInverseTable[Byte(T1[2] shr 16)]; W3 := LastInverseTable[Byte(T1[1]
shr 24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[0];
  W0 := LastInverseTable[Byte(T1[1])]; W1 := LastInverseTable[Byte(T1[0] shr
8)];
  W2 := LastInverseTable[Byte(T1[3] shr 16)]; W3 := LastInverseTable[Byte(T1[2]
shr 24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[1];
  W0 := LastInverseTable[Byte(T1[2])]; W1 := LastInverseTable[Byte(T1[1] shr
8)];
  W2 := LastInverseTable[Byte(T1[0] shr 16)]; W3 := LastInverseTable[Byte(T1[3]
shr 24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[2];
  W0 := LastInverseTable[Byte(T1[3])]; W1 := LastInverseTable[Byte(T1[2] shr
8)];
  W2 := LastInverseTable[Byte(T1[1] shr 16)]; W3 := LastInverseTable[Byte(T1[0]
shr 24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[3];
  // кінець роботи алгоритму
  PLongWord(@OutBuf[0])^ := T0[0]; PLongWord(@OutBuf[4])^ := T0[1];
  PLongWord(@OutBuf[8])^ := T0[2]; PLongWord(@OutBuf[12])^ := T0[3];
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey192;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer);
var
  T0, T1: array [0..3] of longword;
  W0, W1, W2, W3: longword;
begin
  // ініціалізація
  T0[0] := PLongWord(@InBuf[0])^ xor Key[48];
  T0[1] := PLongWord(@InBuf[4])^ xor Key[49];
  T0[2] := PLongWord(@InBuf[8])^ xor Key[50];
  T0[3] := PLongWord(@InBuf[12])^ xor Key[51];
  // Попередня трансформація  11 разів
  // раунд 1
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[44];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[45];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[46];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[47];
  // раунд 2
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
```

```
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[40];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[41];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[42];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[43];
  // раунд 3
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[36];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[37];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[38];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[39];
  // раунд 4
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[32];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[33];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[34];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[35];
  // раунд 5
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[28];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[29];
```

```
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[30];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[31];
  // раунд 6
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[24];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[25];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[26];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[27];
  // раунд 7
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[20];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[21];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[22];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[23];
  // раунд 8
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[16];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[17];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[18];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
```

```
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[19];
  // раунд 9
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[12];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[13];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[14];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[15];
  // раунд 10
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[8];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[9];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[10];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[11];
  // раунд 11
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[4];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[5];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[6];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[7];
  // останій раунд перетворень
```

```
  W0 := LastInverseTable[Byte(T1[0])]; W1 := LastInverseTable[Byte(T1[3] shr
8)];
  W2 := LastInverseTable[Byte(T1[2] shr 16)]; W3 := LastInverseTable[Byte(T1[1]
shr 24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[0];
  W0 := LastInverseTable[Byte(T1[1])]; W1 := LastInverseTable[Byte(T1[0] shr
8)];
  W2 := LastInverseTable[Byte(T1[3] shr 16)]; W3 := LastInverseTable[Byte(T1[2]
shr 24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[1];
  W0 := LastInverseTable[Byte(T1[2])]; W1 := LastInverseTable[Byte(T1[1] shr
8)];
  W2 := LastInverseTable[Byte(T1[0] shr 16)]; W3 := LastInverseTable[Byte(T1[3]
shr 24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[2];
  W0 := LastInverseTable[Byte(T1[3])]; W1 := LastInverseTable[Byte(T1[2] shr
8)];
  W2 := LastInverseTable[Byte(T1[1] shr 16)]; W3 := LastInverseTable[Byte(T1[0]
shr 24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[3];
  // кінець роботи алгоритму
  PLongWord(@OutBuf[0])^ := T0[0]; PLongWord(@OutBuf[4])^ := T0[1];
  PLongWord(@OutBuf[8])^ := T0[2]; PLongWord(@OutBuf[12])^ := T0[3];
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_(const InBuf:
T_Intel_Core_i9_10900K_AES_Buffer; const Key:
T_Intel_Core_i9_10900K_AES_ExpandedKey256;
  var OutBuf: T_Intel_Core_i9_10900K_AES_Buffer);
var
  T0, T1: array [0..3] of longword;
  W0, W1, W2, W3: longword;
begin
  // ініціалізація
  T0[0] := PLongWord(@InBuf[0])^ xor Key[56];
  T0[1] := PLongWord(@InBuf[4])^ xor Key[57];
  T0[2] := PLongWord(@InBuf[8])^ xor Key[58];
  T0[3] := PLongWord(@InBuf[12])^ xor Key[59];
  // Попередня трансформація  13 разів
  // раунд 1
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[52];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[53];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[54];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[55];
  // раунд 2
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
```

```
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[48];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[49];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[50];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[51];
  // раунд 3
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[44];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[45];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[46];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[47];
  // раунд 4
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[40];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[41];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[42];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[43];
  // раунд 5
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[36];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[37];
```

```
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[38];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[39];
  // раунд 6
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[32];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[33];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[34];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[35];
  // раунд 7
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[28];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[29];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[30];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[31];
  // раунд 8
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[24];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[25];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[26];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
```

```
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[27];
  // раунд 9
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[20];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[21];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[22];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[23];
  // раунд 10
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[16];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[17];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[18];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[19];
  // раунд 11
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[12];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[13];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[14];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[15];
  // раунд 12
  W0 := InverseTable[Byte(T1[0])]; W1 := InverseTable[Byte(T1[3] shr 8)];
```

```
  W2 := InverseTable[Byte(T1[2] shr 16)]; W3 := InverseTable[Byte(T1[1] shr
24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[8];
  W0 := InverseTable[Byte(T1[1])]; W1 := InverseTable[Byte(T1[0] shr 8)];
  W2 := InverseTable[Byte(T1[3] shr 16)]; W3 := InverseTable[Byte(T1[2] shr
24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[9];
  W0 := InverseTable[Byte(T1[2])]; W1 := InverseTable[Byte(T1[1] shr 8)];
  W2 := InverseTable[Byte(T1[0] shr 16)]; W3 := InverseTable[Byte(T1[3] shr
24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[10];
  W0 := InverseTable[Byte(T1[3])]; W1 := InverseTable[Byte(T1[2] shr 8)];
  W2 := InverseTable[Byte(T1[1] shr 16)]; W3 := InverseTable[Byte(T1[0] shr
24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[11];
  // раунд 13
  W0 := InverseTable[Byte(T0[0])]; W1 := InverseTable[Byte(T0[3] shr 8)];
  W2 := InverseTable[Byte(T0[2] shr 16)]; W3 := InverseTable[Byte(T0[1] shr
24)];
  T1[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[4];
  W0 := InverseTable[Byte(T0[1])]; W1 := InverseTable[Byte(T0[0] shr 8)];
  W2 := InverseTable[Byte(T0[3] shr 16)]; W3 := InverseTable[Byte(T0[2] shr
24)];
  T1[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[5];
  W0 := InverseTable[Byte(T0[2])]; W1 := InverseTable[Byte(T0[1] shr 8)];
  W2 := InverseTable[Byte(T0[0] shr 16)]; W3 := InverseTable[Byte(T0[3] shr
24)];
  T1[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[6];
  W0 := InverseTable[Byte(T0[3])]; W1 := InverseTable[Byte(T0[2] shr 8)];
  W2 := InverseTable[Byte(T0[1] shr 16)]; W3 := InverseTable[Byte(T0[0] shr
24)];
  T1[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[7];
  // останій раунд перетворень
  W0 := LastInverseTable[Byte(T1[0])]; W1 := LastInverseTable[Byte(T1[3] shr
8)];
  W2 := LastInverseTable[Byte(T1[2] shr 16)]; W3 := LastInverseTable[Byte(T1[1]
shr 24)];
  T0[0] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[0];
  W0 := LastInverseTable[Byte(T1[1])]; W1 := LastInverseTable[Byte(T1[0] shr
8)];
  W2 := LastInverseTable[Byte(T1[3] shr 16)]; W3 := LastInverseTable[Byte(T1[2]
shr 24)];
  T0[1] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[1];
  W0 := LastInverseTable[Byte(T1[2])]; W1 := LastInverseTable[Byte(T1[1] shr
8)];
  W2 := LastInverseTable[Byte(T1[0] shr 16)]; W3 := LastInverseTable[Byte(T1[3]
shr 24)];
  T0[2] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[2];
  W0 := LastInverseTable[Byte(T1[3])]; W1 := LastInverseTable[Byte(T1[2] shr
8)];
  W2 := LastInverseTable[Byte(T1[1] shr 16)]; W3 := LastInverseTable[Byte(T1[0]
shr 24)];
  T0[3] := (W0 xor ((W1 shl 8) or (W1 shr 24)) xor ((W2 shl 16) or (W2 shr 16))
    xor ((W3 shl 24) or (W3 shr 8))) xor Key[3];
  // кінець роботи алгоритму
  PLongWord(@OutBuf[0])^ := T0[0]; PLongWord(@OutBuf[4])^ := T0[1];
  PLongWord(@OutBuf[8])^ := T0[2]; PLongWord(@OutBuf[12])^ := T0[3];
end;
```

```
// Поток раундів шифрування (ECB режим)

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Count, ExpandedKey, Dest);
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Count, ExpandedKey, Dest);
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key256; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Count, ExpandedKey, Dest);
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128; Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
  if Count > 0 then
  begin
    Done := Source.Read(TempIn, Count);
    if Done < Count then
      raise EStreamError.Create(SReadError);
    FillChar(TempIn[Count], SizeOf(TempIn) - Count, 0);
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
  end;
```

```
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192; Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
  if Count > 0 then
  begin
    Done := Source.Read(TempIn, Count);
    if Done < Count then
      raise EStreamError.Create(SReadError);
    FillChar(TempIn[Count], SizeOf(TempIn) - Count, 0);
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
  end;
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256; Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
  if Count > 0 then
  begin
    Done := Source.Read(TempIn, Count);
    if Done < Count then
```

```
      raise EStreamError.Create(SReadError);
    FillChar(TempIn[Count], SizeOf(TempIn) - Count, 0);
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
  end;
end;

// Поток раундів дешифрування (ECB режим)

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(Key, ExpandedKey);
  Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Count, ExpandedKey, Dest);
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128; Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  if (Count mod SizeOf(T_Intel_Core_i9_10900K_AES_Buffer)) > 0 then
    raise E_Intel_Core_i9_10900K_AES_Error.Create(SInvalidInBufSize);
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    Decrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(Key, ExpandedKey);
  Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Count, ExpandedKey, Dest);
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192; Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
```

```
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  if (Count mod SizeOf(T_Intel_Core_i9_10900K_AES_Buffer)) > 0 then
    raise E_Intel_Core_i9_10900K_AES_Error.Create(SInvalidInBufSize);
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    Decrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key256; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(Key, ExpandedKey);
  Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Count, ExpandedKey, Dest);
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256; Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  if (Count mod SizeOf(T_Intel_Core_i9_10900K_AES_Buffer)) > 0 then
    raise E_Intel_Core_i9_10900K_AES_Error.Create(SInvalidInBufSize);
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    Decrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
end;

// Поток раундів шифрування (CBC режим)

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
```

```
  Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source, Count, ExpandedKey,
InitVector, Dest);
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream);
var
  TempIn, TempOut, Vector: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  Vector := InitVector;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    PLongWord(@TempIn[0])^ := PLongWord(@TempIn[0])^ xor PLongWord(@Vector[0])^;
    PLongWord(@TempIn[4])^ := PLongWord(@TempIn[4])^ xor PLongWord(@Vector[4])^;
    PLongWord(@TempIn[8])^ := PLongWord(@TempIn[8])^ xor PLongWord(@Vector[8])^;
    PLongWord(@TempIn[12])^ := PLongWord(@TempIn[12])^ xor
PLongWord(@Vector[12])^;
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Vector := TempOut;
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
  if Count > 0 then
  begin
    Done := Source.Read(TempIn, Count);
    if Done < Count then
      raise EStreamError.Create(SReadError);
    FillChar(TempIn[Count], SizeOf(TempIn) - Count, 0);
    PLongWord(@TempIn[0])^ := PLongWord(@TempIn[0])^ xor PLongWord(@Vector[0])^;
    PLongWord(@TempIn[4])^ := PLongWord(@TempIn[4])^ xor PLongWord(@Vector[4])^;
    PLongWord(@TempIn[8])^ := PLongWord(@TempIn[8])^ xor PLongWord(@Vector[8])^;
    PLongWord(@TempIn[12])^ := PLongWord(@TempIn[12])^ xor
PLongWord(@Vector[12])^;
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
  end;
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source, Count, ExpandedKey,
InitVector, Dest);
end;
```

```
procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream);
var
  TempIn, TempOut, Vector: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  Vector := InitVector;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    PLongWord(@TempIn[0])^ := PLongWord(@TempIn[0])^ xor PLongWord(@Vector[0])^;
    PLongWord(@TempIn[4])^ := PLongWord(@TempIn[4])^ xor PLongWord(@Vector[4])^;
    PLongWord(@TempIn[8])^ := PLongWord(@TempIn[8])^ xor PLongWord(@Vector[8])^;
    PLongWord(@TempIn[12])^ := PLongWord(@TempIn[12])^ xor
PLongWord(@Vector[12])^;
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Vector := TempOut;
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
  if Count > 0 then
  begin
    Done := Source.Read(TempIn, Count);
    if Done < Count then
      raise EStreamError.Create(SReadError);
    FillChar(TempIn[Count], SizeOf(TempIn) - Count, 0);
    PLongWord(@TempIn[0])^ := PLongWord(@TempIn[0])^ xor PLongWord(@Vector[0])^;
    PLongWord(@TempIn[4])^ := PLongWord(@TempIn[4])^ xor PLongWord(@Vector[4])^;
    PLongWord(@TempIn[8])^ := PLongWord(@TempIn[8])^ xor PLongWord(@Vector[8])^;
    PLongWord(@TempIn[12])^ := PLongWord(@TempIn[12])^ xor
PLongWord(@Vector[12])^;
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
  end;
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key256; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForEncryption(Key, ExpandedKey);
  Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source, Count, ExpandedKey,
InitVector, Dest);
end;

procedure Encrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream);
```

```
var
  TempIn, TempOut, Vector: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  Vector := InitVector;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError.Create(SReadError);
    PLongWord(@TempIn[0])^ := PLongWord(@TempIn[0])^ xor PLongWord(@Vector[0])^;
    PLongWord(@TempIn[4])^ := PLongWord(@TempIn[4])^ xor PLongWord(@Vector[4])^;
    PLongWord(@TempIn[8])^ := PLongWord(@TempIn[8])^ xor PLongWord(@Vector[8])^;
    PLongWord(@TempIn[12])^ := PLongWord(@TempIn[12])^ xor
PLongWord(@Vector[12])^;
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
    Vector := TempOut;
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
  if Count > 0 then
  begin
    Done := Source.Read(TempIn, Count);
    if Done < Count then
      raise EStreamError.Create(SReadError);
    FillChar(TempIn[Count], SizeOf(TempIn) - Count, 0);
    PLongWord(@TempIn[0])^ := PLongWord(@TempIn[0])^ xor PLongWord(@Vector[0])^;
    PLongWord(@TempIn[4])^ := PLongWord(@TempIn[4])^ xor PLongWord(@Vector[4])^;
    PLongWord(@TempIn[8])^ := PLongWord(@TempIn[8])^ xor PLongWord(@Vector[8])^;
    PLongWord(@TempIn[12])^ := PLongWord(@TempIn[12])^ xor
PLongWord(@Vector[12])^;
    Encrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError.Create(SWriteError);
  end;
end;

// Поток раундів дешифрування (CBC режим)

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key128; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(Key, ExpandedKey);
  Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source, Count, ExpandedKey,
InitVector, Dest);
end;

procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey128;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Vector1, Vector2: T_Intel_Core_i9_10900K_AES_Buffer;
```

```
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  if (Count mod SizeOf(T_Intel_Core_i9_10900K_AES_Buffer)) > 0 then
    raise E_Intel_Core_i9_10900K_AES_Error.Create(SInvalidInBufSize);
  Vector1 := InitVector;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError(SReadError);
    Vector2 := TempIn;
    Decrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    PLongWord(@TempOut[0])^ := PLongWord(@TempOut[0])^ xor
PLongWord(@Vector1[0])^;
    PLongWord(@TempOut[4])^ := PLongWord(@TempOut[4])^ xor
PLongWord(@Vector1[4])^;
    PLongWord(@TempOut[8])^ := PLongWord(@TempOut[8])^ xor
PLongWord(@Vector1[8])^;
    PLongWord(@TempOut[12])^ := PLongWord(@TempOut[12])^ xor
PLongWord(@Vector1[12])^;
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError(SWriteError);
    Vector1 := Vector2;
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
end;


procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key192; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(Key, ExpandedKey);
  Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source, Count, ExpandedKey,
InitVector, Dest);
end;


procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey192;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Vector1, Vector2: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  if (Count mod SizeOf(T_Intel_Core_i9_10900K_AES_Buffer)) > 0 then
    raise E_Intel_Core_i9_10900K_AES_Error.Create(SInvalidInBufSize);
  Vector1 := InitVector;
```

```
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do


begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError(SReadError);
    Vector2 := TempIn;
    Decrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    PLongWord(@TempOut[0])^ := PLongWord(@TempOut[0])^ xor
PLongWord(@Vector1[0])^;
    PLongWord(@TempOut[4])^ := PLongWord(@TempOut[4])^ xor
PLongWord(@Vector1[4])^;
    PLongWord(@TempOut[8])^ := PLongWord(@TempOut[8])^ xor
PLongWord(@Vector1[8])^;
    PLongWord(@TempOut[12])^ := PLongWord(@TempOut[12])^ xor
PLongWord(@Vector1[12])^;
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError(SWriteError);
    Vector1 := Vector2;
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
end;


procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const Key: T_Intel_Core_i9_10900K_AES_Key256; const InitVector:
T_Intel_Core_i9_10900K_AES_Buffer; Dest: TStream);
var
  ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;
begin
  Expand_Intel_Core_i9_10900K_AES_KeyForDecryption(Key, ExpandedKey);
  Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source, Count, ExpandedKey,
InitVector, Dest);
end;


procedure Decrypt_Intel_Core_i9_10900K_AES_StreamCBC(Source: TStream; Count:
cardinal;
  const ExpandedKey: T_Intel_Core_i9_10900K_AES_ExpandedKey256;  const
InitVector: T_Intel_Core_i9_10900K_AES_Buffer;
  Dest: TStream);
var
  TempIn, TempOut: T_Intel_Core_i9_10900K_AES_Buffer;
  Vector1, Vector2: T_Intel_Core_i9_10900K_AES_Buffer;
  Done: cardinal;
begin
  if Count = 0 then
  begin
    Source.Position := 0;
    Count := Source.Size;
  end
  else Count := Min(Count, Source.Size - Source.Position);
  if Count = 0 then exit;
  if (Count mod SizeOf(T_Intel_Core_i9_10900K_AES_Buffer)) > 0 then
    raise E_Intel_Core_i9_10900K_AES_Error.Create(SInvalidInBufSize);
  Vector1 := InitVector;
  while Count >= SizeOf(T_Intel_Core_i9_10900K_AES_Buffer) do
  begin
    Done := Source.Read(TempIn, SizeOf(TempIn));
    if Done < SizeOf(TempIn) then
      raise EStreamError(SReadError);
    Vector2 := TempIn;
    Decrypt_Intel_Core_i9_10900K_AES_(TempIn, ExpandedKey, TempOut);
    PLongWord(@TempOut[0])^ := PLongWord(@TempOut[0])^ xor
PLongWord(@Vector1[0])^;
```

```
    PLongWord(@TempOut[4])^ := PLongWord(@TempOut[4])^ xor
PLongWord(@Vector1[4])^;
    PLongWord(@TempOut[8])^ := PLongWord(@TempOut[8])^ xor
PLongWord(@Vector1[8])^;
    PLongWord(@TempOut[12])^ := PLongWord(@TempOut[12])^ xor
PLongWord(@Vector1[12])^;
    Done := Dest.Write(TempOut, SizeOf(TempOut));
    if Done < SizeOf(TempOut) then
      raise EStreamError(SWriteError);
    Vector1 := Vector2;
    Dec(Count, SizeOf(T_Intel_Core_i9_10900K_AES_Buffer));
  end;
end;

end.
```

Файл Main.pas – основна програма

```
unit Main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, El_Intel_Core_i9_10900K_AES_, Math, Buttons;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    OpenDialog1: TOpenDialog;
    Button1: TButton;
    Button2: TButton;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Edit2: TEdit;
    Label_Time: TLabel;
    Label7: TLabel;
    Label8: TLabel;
    Label9: TLabel;
    Label_Status: TLabel;
    Memo_PlainText: TMemo;
    Memo_CyperText: TMemo;
    Label11: TLabel;
    Label12: TLabel;
    Memo_UncipherText: TMemo;
    Label13: TLabel;
    BitBtn_Encrypt: TBitBtn;
    BitBtn_Decypt: TBitBtn;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure BitBtn_EncryptClick(Sender: TObject);
    procedure BitBtn_DecyptClick(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;


var
  Form1: TForm1;
  EncryptedText: string;

function StringToHex(S: string): string; forward;
function HexToString(S: string): string; forward;

implementation

uses about;

{$R *.DFM}

function StringToHex(S: string): string;
var
      i: integer;

begin
```

```
  Result := '';

  // послідовно беремо одиничні символи та перетворюємо їх
  // до шістнадцяткових...
      for i := 1 to Length( S ) do
      Result := Result + IntToHex( Ord( S[i] ), 2 );
end;

function HexToString(S: string): string;
var
      i: integer;

begin
  Result := '';

  // послідовно беремо одиничні шістнадцяткові символи та перетворюємо їх
  // ASCII символів...
  for i := 1 to Length( S ) do
  begin
    // Туту обробляється тільки 2 шістнадцяткових блока...
      if ((i mod 2) = 1) then
            Result := Result + Chr( StrToInt( '0x' + Copy( S, i, 2 )));
  end;
end;


procedure TForm1.Button1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
    Edit1.Text := OpenDialog1.FileName;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  Source, Dest: TFileStream;
  SrcFile, DestFile: string;
  Start, Stop: cardinal;
  Size: integer;
  Key: T_Intel_Core_i9_10900K_AES_Key128;
  SrcBuf, DstBuf: array [0..16383] of byte;
  SrcSize, DstSize: integer;
begin
  // Шифрування
  Label_Status.Caption := 'Шифруємо...';
  Refresh;
  Source := TFileStream.Create(Edit1.Text, fmOpenRead);
  try
    Label4.Caption := IntToStr(Source.Size div 1024) + ' KB';
    Refresh;
    DestFile := ExtractFilePath(Application.ExeName) + 'aestemp.enc';
    Dest := TFileStream.Create(DestFile, fmCreate);
    try
      Size := Source.Size;
      Dest.WriteBuffer(Size, SizeOf(Size));

      FillChar(Key, SizeOf(Key), 0);
      Move(PChar(Edit2.Text)^, Key, Min(SizeOf(Key), Length(Edit2.Text)));

      Start := GetTickCount;
      Encrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, 0, Key, Dest);
      Stop := GetTickCount;
      Label_Time.Caption := IntToStr(Stop - Start) + ' ms';
      Refresh;
    finally
      Dest.Free;
    end;
  finally
    Source.Free;
  end;
```

```
  // Дешифрування
  Label_Status.Caption := 'Дешифруємо...';
  Refresh;
  Source := TFileStream.Create(DestFile, fmOpenRead);
  try
    Source.ReadBuffer(Size, SizeOf(Size));
    SrcFile := ExtractFilePath(Application.ExeName) + 'aestemp.dec';
    Dest := TFileStream.Create(SrcFile, fmCreate);
    try
      Start := GetTickCount;
      Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Source.Size -
Source.Position, Key, Dest);
      Dest.Size := Size;
      Stop := GetTickCount;
      Label8.Caption := IntToStr(Stop - Start) + ' ms';
      Refresh;
    finally
      Dest.Free;
    end;
  finally
    Source.Free;
  end;
  // Порівнюємо
  Label_Status.Caption := 'Порівнюємо...';
  Refresh;
  Source := TFileStream.Create(Edit1.Text, fmOpenRead);
  SrcSize := Source.Size;
  try
    Dest := TFileStream.Create(SrcFile, fmOpenRead);
    DstSize := Dest.Size;
    try
      repeat
        Source.ReadBuffer(SrcBuf, Min(SizeOf(SrcBuf), SrcSize));
        Dest.ReadBuffer(DstBuf, Min(SizeOf(DstBuf), DstSize));
        if not CompareMem(@SrcBuf, @DstBuf, Max(Min(SizeOf(DstBuf), DstSize),
Min(SizeOf(SrcBuf), SrcSize))) then
        begin
          ShowMessage('ПОМИЛКА: файл було змінено!!!');
          DeleteFile(SrcFile);
          DeleteFile(DestFile);
          exit;
        end;
        Dec(SrcSize, Min(SizeOf(SrcBuf), SrcSize));
        Dec(DstSize, Min(SizeOf(DstBuf), DstSize));
      until (SrcSize = 0) or (DstSize = 0);
      ShowMessage('Зміни не знайдено');
    finally
      Dest.Free;
    end;
  finally
    Source.Free;
  end;
  Label_Status.Caption := 'Видалено...';
  Refresh;
  Label_Status.Caption := '';
end;


(*****************************************************************************
   Використовуємо TStringStream... для шифрування
*****************************************************************************)
procedure TForm1.BitBtn_EncryptClick(Sender: TObject);
var
  Source: TStringStream;
  Dest: TStringStream;
  Start, Stop: cardinal;
  Size: integer;
  Key: T_Intel_Core_i9_10900K_AES_Key128;
```

```
begin
  // Шифрування
  Label_Status.Caption := 'Шифруємо...';
  Refresh;
  Source := TStringStream.Create( Memo_PlainText.Text );
  Dest   := TStringStream.Create( '' );

  try
    // Зберігаємо дані до пам'яті...
    Size := Source.Size;
    Dest.WriteBuffer( Size, SizeOf(Size) );

    // Початковий ключ...
    FillChar( Key, SizeOf(Key), 0 );
    Move( PChar(Edit2.Text)^, Key, Min( SizeOf( Key ), Length( Edit2.Text ) ));

    // Починаємо шифрування...
    Починаємо := GetTickCount;
    Encrypt_Intel_Core_i9_10900K_AES_StreamECB( Source, 0, Key, Dest );
    Stop := GetTickCount;
    Label_Time.Caption := IntToStr(Stop - Start) + ' ms';
    Refresh;

    // Видаємо зашифрований текст на екран...
        Memo_CyperText.Lines.BeginUpdate;
    Memo_CyperText.Text := StringToHex( Dest.DataString );
        Memo_CyperText.Lines.EndUpdate;

  finally
    Source.Free;
    Dest.Free;
  end;
end;

procedure TForm1.BitBtn_DecyptClick(Sender: TObject);
var
  Source: TStringStream;
  Dest: TStringStream;
  Start, Stop: cardinal;
  Size: integer;
  Key: T_Intel_Core_i9_10900K_AES_Key128;
  EncryptedText: TStrings;
  S: string;

begin
  // перетворюємо шістнадцяткове число до рядка перед дешифруванням...
  Source := TStringStream.Create( HexToString( Memo_CyperText.Text ));
  Dest   := TStringStream.Create( '' );

  try
    // Починаємо дешифрування...
    Size := Source.Size;
    Починаємо := GetTickCount;
    Source.ReadBuffer(Size, SizeOf(Size));

    // Початковий ключ...
    FillChar(Key, SizeOf(Key), 0);
    Move(PChar(Edit2.Text)^, Key, Min(SizeOf(Key), Length(Edit2.Text)));

    // Дешифруємо...
    Decrypt_Intel_Core_i9_10900K_AES_StreamECB(Source, Source.Size -
Source.Position, Key, Dest);
    Stop := GetTickCount;
    Label8.Caption := IntToStr(Stop - Start) + ' ms';
    Refresh;

    // Виводимо розшифрований текст...
    Memo_UncipherText.Text := Dest.DataString;
```

```
  finally
    Source.Free;
    Dest.Free;
  end;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
FormAbout.Show;
end;

end.
```

Файл about.pas - довідка

```
unit about;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, jpeg, ExtCtrls;

type
  TForm5 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Image1: TImage;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form5: TForm5;

implementation

{$R *.dfm}

procedure TForm5.FormCreate(Sender: TObject);
begin
Memo1.Clear;
Memo1.Lines.Add(' БАКАЛАВРСЬКИЙ ПРОЕКТ');
Memo1.Lines.Add('');
Memo1.Lines.Add(' на  тему: ');
Memo1.Lines.Add('');
Memo1.Lines.Add(' Програмне забезпечення системи кібербезпеки для забезпечення
захисту інформації з використанням спеціальних інструкцій Intel Core i9-
10900K');
Memo1.Lines.Add('');
Memo1.Lines.Add(' Керівник: Смірнов С.А.');
Memo1.Lines.Add('');
Memo1.Lines.Add(' Розробив: студент Степаненко Євгеній Олександрович');
Memo1.Lines.Add('                  гр. КБ-19');
Memo1.Lines.Add('');
Memo1.Lines.Add('Кропивницький 2023');
Memo1.Lines.Add('');
end;

procedure TForm5.Button1Click(Sender: TObject);
begin
Form5.Close;
end;
end.
```