МІНІСТЕРСТВО ОСВІТИ І НАУКИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

На правах рукопису
УДК 681.3.06:004.056.5 (043.3)

Навроцький Денис Олександрович

МЕТОДИ ПОБУДОВИ СИМЕТРИЧНИХ КРИПТОГРАФІЧНИХ ШИФРІВ З ВИКОРИСТАННЯМ ТРИВИМІРНИХ ПЕРЕТВОРЕНЬ

Спеціальність 05.13.21 – Системи захисту інформації

Дисертація на здобуття наукового ступеня
кандидата технічних наук

Науковий керівник
Білецький Анатолій Якович
доктор технічних наук., професор

Київ – 2017

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, СИМВОЛІВ І ТЕРМІНІВ

AES – (Advanced Encryption Standard) – розширений стандарт шифрування;

CBC – (Cipher Block Chaining) – режим зчеплення блоків шифротекста;

DES – (Data Encryption Standard) – симетричний алгоритм шифрування;

ECB – (Electronic Codebook) – режим електронної кодової книги;

FIPS – (Federal Information Processing Standards) федеральні стандарти США;

MARS – шифр-кандидат в AES;

NIST – пакет статистичних тестів, розроблений Національним інститутом стандартів і технологій США;

OFB – (Output Feedback) – режим зворотного зв'язку по виходу;

RC6 – симетричний блоковий криптографічний алгоритм;

RSB – (Round Step Block) – розроблюваний симетричний алгоритм шифрування;

АКК – арифметичне «ковзне кодування»;

БПЛА – безпілотний літальний апарат;

КЗІ – криптографічний захист інформації;

ЛЗЗ – лінійні зворотні зв'язки;

ЛРС – лінійні регістри зсуву;

ЛКК – логічне «ковзне кодування»;

НП – незвідні поліноми;

ПКК – примітиви «ковзне кодування»;

ПрМ – примітивні матриці;

ПрП – примітивні поліноми;

СБШ – симетричний блоковий шифратор;

КК – «ковзне кодування»;

ЗКК – змішане «ковзне кодування»;

ВСТУП

Актуальність. Обчислювальні системи постійно вдосконалюються і набувають нових функціональних властивостей, до яких належать можливості розподілених паралельних розрахунків, хмарні технології, просторові (графічні) розрахунки тощо. З розвитком електроніки і програмування зростають ризики в криптографічному захисті. Проведені конкурси з обрання стандартів шифрування AES, NESSIE, CRYPTREC тощо були розраховані на апаратні та програмні можливості того часу. На сьогодні доволі поширені на ринку шифри з репутацією стійких і швидких, такі як AES, ГОСТ 28147-89, ДСТУ 7624:2014, 3DES та ін. Ці шифри намагаються адаптувати під сучасні потреби захисту даних. З поширенням технологій інтернет-речей і безпілотних літальних апаратів (БПЛА) постала проблема захищеності даних малою обчислювальною потужністю. В комп'ютерах гострої проблеми зі швидкодією й об'ємами пам'яті не існує, проте в мікроконтролерах ця проблема актуальна. Оскільки будь-які розрахунки займають багато часу і споживають енергію, для автономних систем, які живляться від акумуляторів, це вкрай критично. Наприклад БПЛА, у якому на одному мікроконтролері має бути реалізована як навігаційна система, так і шифрування даних для командної, телеметричної, відеоінформації. Адаптація існуючих шифрів під нові задачі не є тривіальною і її вирішують за допомогою сторонніх фірм-розробників, які досить часто використовують закритий код і *know-how* принцип. Це спричиняє виникнення шпаринок (англ. *backdoor*) у реалізаціях шифрів. Отже, слід зазначити відмінність між криптографічною стійкістю алгоритму й апаратно-програмною реалізацією цього алгоритму на конкретному обладнанні. Останнім часом склалась така ситуація, що під брендом відомих шифрів на ринку наявні безліч вразливих систем шифрування. Це підтверджується тим, що досить часто виробники шифрувального обладнання заявляють, що мають можливість відновити зашифровані дані на їх обладнанні у випадку втрати ключа

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

Значний внесок у розвиток криптографічних методів захисту інформації внесли такі відомі вчені, як І.Д. Горбенко, В.І. Долгов, В.К. Задірака, І.Л. Єрош, М.А. Іванов, Л.В. Ковальчук, Г.В. Кузнецов, О.О. Кузнецов, М.Е. Масленніков, А.А. Молдовян, Н.А. Молдовян, А.М. Олексійчук, А.А. Петров, А.Г. Ростовцев, Ю.С. Харін та ін. З-поміж закордонних науковців варто згадати таких: А. Бірюков, Е. Біхам, Й. Даймен, В. Діффі, Л. Кнудсен, Н. Кобліц, М. Мацуі, Дж. Мессі, Н. Смарт, Р. Смит, В. Столінгс, Р. Рівест, В. Реймен, Х. К. А. ван Тілборг, Х. Фейстель, Н. Фергюсон, А. Шамір, Б. Шнаєр, М. Хеллман та ін.

Переважна більшість досліджень, орієнтовані на методи, які ускладнюють розрахунки, тобто йдуть шляхом збільшення розрахунків. Такий підхід спричиняє великі витрати машинного часу і відповідно споживає багато енергії.

Таким чином, розроблення і дослідження методів швидких криптографічних перетворень для систем з малою швидкодією й обмеженою кількістю пам’яті є актуальними науковими завданнями, які мають теоретичне і практичне значення.

Зв’язок роботи з науковими програмами, планами, темами.

Тематика дисертаційної роботи та отримані результати безпосередньо пов’язані із:

1. Концепцією інформаційної безпеки України;

2. Концепцією розвитку сектору безпеки й оборони України;

3. Указом Президента України № 92/2016 від 4 березня 2016 року «Про Стратегію кібербезпеки України»;

4. «Основними науковими напрямами та найважливішими проблемами фундаментальних досліджень у галузі природничих, технічних і гуманітарних наук НАН України на 2014–2018 роки» в частині п.1.2.8.1 «Розробка методів та

інформаційних технологій розв’язання задач комп’ютерної криптографії та стеганографії»;

5. Стратегією національної безпеки України від 26 травня 2015 року № 287/2015 у контексті п.4.12 «Забезпечення кібербезпеки і безпеки інформаційних ресурсів, зокрема реформування системи технічного і криптографічного захисту інформації з урахуванням практики держав-членів НАТО та ЄС з досліджень та інновацій;

6. «Горизонт-2020», зокрема за напрямами DS-05-2016 та DS-06-2017 («Визначення нових напрямів інноваційних наукових досліджень в Європі щодо забезпечення кібербезпеки як відповідь на нові тенденції і передові технології»);

7. НДР «Розробка та впровадження програмних засобів захисту інформації від несанкціонованого доступу в електронних системах документообігу у вищих навчальних закладах України» (2010–2011 рр., номер державної реєстрації 0110U000222).

Мета і задачі дослідження. Мета дисертаційної роботи полягає у підвищенні ефективності (стійкості, швидкодії, зменшення ресурсоємності) криптографічного захисту інформації на основі застосування нових блокових і потокових шифрів з використанням динамічно керованих тривимірних криптографічних примітивів.

Для досягнення поставленої мети необхідно розв’язати такі основні завдання:

- провести аналіз сучасних криптографічних методів захисту інформації за критеріями стійкості, швидкодії та необхідних ресурсів для розрахунків, а також обґрунтувати шляхи вдосконалення алгоритмів симетричного шифрування;

- розробити методи формування динамічно керованих примітивів лінійного розсіювання, нелінійної заміни та «ковзного кодування» на основі узагальнених перетворень Грея та матриць Галуа для тривимірного простору;

- на основі запропонованих криптографічних примітивів розробити методи криптографічної обробки даних (блокового, потокового) тривимірного шифрування даних з метою підвищення ефективності (стійкості, швидкодії, менш ресурсоємні) криптографічного захисту інформації;

- розробити апаратно-програмне забезпечення і провести лабораторні випробування засобів захисту командно-телеметричної (КТ) та відеоінформації (ВІ) в каналах зв'язку наземного пункту керування з безпілотним літальним апаратом (НПК–БПЛА).

Об'єктом дослідження є процес перетворення інформації в симетричних криптографічних шифрах на основі тривимірних примітивів.

Предметом дослідження є методи побудови симетричних криптосистем, що забезпечують підвищення ефективності захисту інформації.

Методи дослідження. Проведені дослідження ґрунтуються на сучасній теорії криптографічного захисту інформації, теорії чисел, алгебричній теорії груп, скінчених полях Галуа, теорії незвідних і примітивних поліномів, а також теорії синтезу генераторів псевдовипадкових послідовностей, що засновані на узагальнених лінійних регістрах зсуву з лінійними зворотними зв'язками, теорії ймовірностей та математичній статистиці, об'єктно-орієнтованого програмування тощо.

Наукова новизна одержаних результатів полягає в такому:

уперше розроблено метод і його модифікації для формування динамічно керованих примітивів лінійного розсіювання, нелінійної заміни та «ковзного кодування» на основі узагальнених перетворень Грея та матриць Галуа для тривимірного простору, які за рахунок розроблених динамічних дискретних математичних моделей на базі операцій над тривимірними матрицями, властивостями невироджених тривимірних матриць у полі Галуа; класичних (*лівосторонніх*) і так званих *правосторонніх* та *складених* кодів та рандомізованих *кодів Грея* в тривимірному просторі дали можливість на їх основі синтезувати та розробити динамічно керовані тривимірні

криптографічні методи побудови перемішування (permutation 3D), нелінійної заміни (substitution 3D), матричного перетворення (matrix 3D), стохастичного циклічного зсуву (shift 3D), «ковзного» кодування 3D (slider code 3D), що дало можливість збільшити швидкодію і стійкість шифрів та зменшити необхідний об’єм пам’яті для роботи шифру;

удосконалено метод синтезу матриць для таблиць підстановки і перестановки на основі запропонованих криптографічних примітивів і розробленого методу криптографічної обробки даних (блокового, потокового) тривимірного шифрування даних, які за рахунок методів синтезу примітивних матриць Галуа і Фібоначчі, сполучених варіантів, над простими полями Галуа характеристики 2, дали можливість розширити множину узагальнених генераторів псевдовипадкових послідовностей, а також запропонувати нові підходи до розв’язання проблеми формування таємних ключів шифрування абонентами мережі з відкритими каналами зв’язку;

отримали подальший розвиток методи розробки засобів захисту командно-телеметричної та відеоінформації в каналах зв’язку наземного пункту керування з безпілотним літальним апаратом, які за рахунок методів симетричного блокового криптографічного перетворення інформації з динамічно керованими параметрами шифрування (криптографічні перетворення виконуються в тривимірному просторі) дали можливість в алгоритмах шифрування здійснювати оперативну модифікацію параметрів криптографічних примітивів при переході до чергового блоку тексту, що перетворюється.

Практичне значення одержаних результатів. Практична цінність роботи полягає в такому:

запропоновано спосіб тривимірних перетворень блокових та потокових шифрів (БШ і ПШ) на основі розроблених методів формування динамічно керованих примітивів лінійного розсіювання, нелінійної заміни та «ковзного кодування» на основі узагальнених перетворень Грея та матриць Галуа для

тривимірного простору для побудови більш швидких і таких, що потребують меншого об’єму пам’яті криптосистем;

розроблені та впроваджені алгоритми симетричних RSB-64-3D блокових і 3DMatrix потокового шифрування інформації, які доведені до рівня програмної реалізації на мові C# і C++ для ПК і мікроконтролера, що дало можливість провести експериментальне дослідження запропонованих рішень для криптографічного захисту командно-телеметричної інформації в каналі зв’язку НПК–БПЛА і дало можливість підвищити швидкодію і зменшити необхідний об’єм пам’яті для забезпечення криптографічного захисту інформації як для мікроконтролерних пристроїв, так і для комп’ютерів;

розроблено ПЗ для статистичного аналізу криптограм на базі тестів NIST STS і DIEHARD, які доповнені власними методами оцінювання шифрограм;

за результатами дисертаційних розробок отримано дев'ять патентів України на «Спосіб криптографічного захисту інформації»;

результати дисертаційної роботи впроваджено у навчальному процесі кафедри електроніки Національного авіаційного університету, кафедри виробництва приладів Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського», та у науково-технічних розробках ТОВ «Агфар», ТОВ «Гратис, Лтд», що підтверджено відповідними актами впровадження.

Достовірність наукових результатів підтверджується коректним використанням методів досліджень та збіжністю результатів машинного експерименту з теоретичними даними, отриманими в результаті досліджень.

Особистий внесок здобувача. Основні положення і результати дисертаційної роботи, що виносяться на захист, отримані автором самостійно. У роботах, написаних у співавторстві, автору належать: [1] – програмне моделювання процесу розпізнавання; [2] – множина секвентних функцій; [3] – алгоритм синхронного потокового шифрування; [5] – синтез і реалізація SCSPS шифратора; [6] – синтез програмно-моделюючого комплексу захисту каналу

зв’язку БПЛА; [7] – досліджено криптографічний примітив нелінійної підстановки у тривимірному просторі для шифру RSB3D; [8] – дослідження та розробка криптографічних примітивів за допомогою примітивних поліномів; [9] – програмна модель Ch-CRC; [11] – метод направленого машинного перебору для синтезу квазіеквідистантних кодів.

Із робіт, опублікованих у співавторстві, у дисертаційній роботі використовуються результати, отримані особисто здобувачем.

Апробація результатів дисертації. Основні положення дисертаційної роботи доповідалися та обговорювалися більш ніж на 10 міжнародних та всеукраїнських наукових конференціях, серед яких: міжнародна науково-технічна конференція «Захист інформації і безпека інформаційних систем» (Львів, 2012); всеукраїнська науково-практична конференція «Проблеми та перспективи розвитку авіації та космонавтики» (Київ, 2012); науково-технічна конференція «Наукоємні технології» (Київ, 2012); міжнародна науково-технічна конференція «ABIA–2013» (Київ, 2013); всеукраїнська науково-практична конференція «Проблеми навігації і управління рухом» (Київ, 2013); міжнародна науково-технічна конференція «Розвиток наукових досліджень 2013» (Полтава, 2013); науково-технічна конференція «Проблеми розвитку глобальної системи зв’язку, навігації, спостереження та організації повітряного руху CNS/ATM» (Київ, 2014); міжнародна науково-технічна конференція «ITSEC» (Київ, 2015); міжнародна науково-технічна конференція «ABIA–2015» (Київ, 2015); науково-практична конференція «Сучасні тенденції розвитку системного програмування» (Київ, 2015); науково-методичних семінарах кафедри електроніки Національного авіаційного університету.

Публікації. Основні положення дисертації опубліковано у 32 наукових працях, у тому числі 1 стаття входить у наукометричну базу даних Scopus, 12 статей – у фахових виданнях України, які входять до міжнародних наукометричних баз даних, а також 9 патентах України на корисну модель та 10 тезах доповідей на конференціях.

Структура роботи та її обсяг. Дисертація складається зі вступу, чотирьох розділів, загальних висновків, додатків, списку використаних джерел, а також має 155 сторінок основного тексту, 81 рисунок, 25 таблиць, 68 сторінок додатків. Список використаних джерел містить 133 найменування і займає 20 сторінок. Загальний обсяг роботи − 243 сторінки.

РОЗДІЛ 1.
АНАЛІЗ СТАНУ СИМЕТРИЧНОГО БЛОЧНОГО ШИФРУВАННЯ ДАНИХ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

На основі аналізу відкритих літературних джерел в даному розділі розглядаються базові поняття шифрування, наведена класифікація та коротка характеристика алгоритмів криптографічного захисту інформації, систематизовані основні атаки на шифри та визначені основні критерії ефективності алгоритмів симетричного блочного шифрування.

1.1. Базові поняття шифрування інформації

Розглянуто базові поняття шифрування, до яких входять: режими шифрування, поширення помилок, вибір режиму шифрування, задачі криптографії, криптографічні примітиви, алгоритми шифрування, ітеративні блочні шифри і їх структура і принципи побудови, стандартизація і застосування блочних шифрів.

Симетричні режими шифрування. Основні режими симетричного шифрування рекомендовані NIST Special Publication 800-38A [12]: Electronic code book (ECB), Cipher block chaining (CBC), Propagating cipher block chaining (PCBC), Cipher feed back (CFB), Output feed back (OFB), Counter mode (CTR). Режими шифрування рекомендовані NIST Special Publication 800-38C[13] і 800-38D[14]: CCM (Counter with CBC), GCM (Galois/Counter Mode). Введені у новому IEEE стандарті 802.1AR «Standard for Local and Metropolitan Area Networks Secure Device Identity»: LRW, XTS. Для блоків великого розміру використовують режими, описані в стандарті SISWG (IEEE P1619): CMS (CBC-mask-CBC), EME (ECB-Mix-ECB).

Поширення помилок. Для будь-якого режиму шифрування помилковий біт в блоці шифротексту призводить до того, що результат його розшифрування

виявляться зіпсованим. В режимі CFB, OFB і CTR зіпсований біт буде мати ту саму позицію в розшифрованому блоці, що і біт помилки в блоці шифротексту, на інші біти блока помилка не поширюється. В режимі ECB і CBC спотвореним може виявитись будь-який біт блоку, з вірогідністю близькою до 50% (залежить від надійності самого шифру). При цьому в режимах ECB, OFB і CTR зіпсованим виявляється виключно блок, який отримано в результаті розшифрування зіпсованого блоку. В режимі CBC помилково розшифровуванний буде також і наступний блок, при цьому зіпсовані біти будуть відповідати бітам помилки в шифротексту попереднього блоку. В режимі CFB біт помилки в сегменті шифротексту впливає на наступні b/s (округлюючи до цілого, b – довжина блоку, s – довжина сегменту) сегментів, а помилковим може виявитись будь-який з біт розшифрованого тексту [19].

Присутність блоків помилок в векторі ініціалізації також спричиняє шкоду процесу розшифрування. В режимі OFB біт помилки в IV вражає кожний блок шифротексту в відповідному повідомленні. В режимі CFB при помилках в векторі ініціалізації зіпсованим виявляється, принаймні, перший сегмент шифротексту. Зіпсованість інших сегментів залежить від положення самого правого біта в IV (в гіршому випадку постраждають b/s сегментів шифротексту). При використанні режимів OFB і CFB в результаті виникнення біта помилки в IV зіпсованим може виявитись будь-який біт пошкодженого шифротексту. В режимі CBC зіпсовані будуть виключно біти першого блоку шифротексту, які знаходяться на позиціях, що відповідають бітам помилки в векторі ініціалізації. Для режиму CTR помилковий біт в значенні лічильника призводить до того, що будь-який біт при розшифруванні відповідного шифротексту може виявитись зіпсованим з вірогідність близькою до 50% [16].

Окрім виникнення помилкового біту в блоці шифротексту може статись стирання або вставка біта. Це призводить до порушення меж усіх наступних блоків шифротексту, а результати розшифрування будуть абсолютно невірними, доки синхронізація меж не відновиться. При використанні режиму

1-бітного CFB синхронізація відновлюється автоматично через $b+1$ позицій після з'явившегося або зниклого біта. В інших режимах автоматичного відновлення синхронізації не відбувається [17].

Вибір режиму шифрування. Вибір режиму шифрування залежить від обраної мети. Для звичайного відкритого тексту можна використовувати CBC, CFB або OFB. Для шифрування файлів краще використовувати ECB або більш сучасні LRW і XTS. Для файлів великого розміру запропоноване використання режимів CMS та EMS.

Задачі криптографії. Криптографія використовується для досягнення наступних цілей:

1) Конфіденційність: захист даних або персональної інформації користувача від несанкціонованого перегляду;
2) Цілісність даних: захист даних від несанкціонованих змін;
3) Автентифікація: перевірка того, що дані надходять від визначеної особи;
4) Невідмовність: жодна сторона не має можливості заперечувати факт відправки повідомлення.

Криптографічні примітиви. Для досягнення цілей криптографії можна використовувати алгоритми і правила, відомі як криптографічні примітиви, для створення криптографічних схем. В таблиці 1.1 нижче наведено криптографічні примітиви і опис їх використання.

Таблиця 1.1

Криптографічні примітиви і їх використання

Криптографічний примітив	Використання
Шифрування з закритим ключем (симетричне шифрування)	Здійснює перетворення даних з метою запобігання їх перегляду третьою стороною. При такому способі шифрування для шифрування і розшифрування даних використовується один спільний закритий ключ.

Продовження таблиці 1.1

Шифрування з відкритим ключем (асиметричне шифрування)	Здійснює перетворення даних з метою запобігання їх перегляду третьою стороною. При такому способі шифрування для шифрування і розшифрування даних використовується набір, що складається з відкритого і закритого ключів.
Створення криптографічного підпису	Дозволяє перевіряти, чи дійсно дані надходять від конкретної особи, за допомогою унікального цифрового підпису цієї особи. Цей процес також використовує хеш-функції.
Криптографічне хешування	Відображає дані будь-якого розміру в байтову послідовність фіксованої довжини. Результати хешування статистично унікальні; послідовність, що відрізняється хоча б одним байтом не буде перетворена в таке саме значення.

CNG (Cryptography Next Generation) [18]**.** CNG надає криптографічні примітиви, що зведені в табл. 1.2. для наступних класів алгоритмів.

Таблиця 1.2

Криптографічні примітиви CNG

Клас алгоритму	Опис
Генератор випадкових чисел (Random number generator)	Змінні генерації випадкових чисел (RNG).
Хешування (Hashing)	Алгоритми хешування, такі як SHA1 або SHA2.
Симетричне шифрування (Symmetric encryption)	Алгоритми симетричного шифрування, такі як AES, 3DES або RC4.
Асиметричне шифрування (Asymmetric encryption)	Асиметричні (з відкритим ключем) які підтримують шифрування, такі як RSA.
Підпис (Signature)	Цифровий підпис такі як DSA або ECDSA. Також використовується в RSA.
Таємна угода (Secret agreement)	Алгоритми таємних угод, такі як Діффі-Хеллмана (DH) і елептичні криві Діффі-Хеллмана (ECDH).

Повна ілюстрація дизайну і функціонування CNG криптографічних примітивів наведена на рис. 1.1.

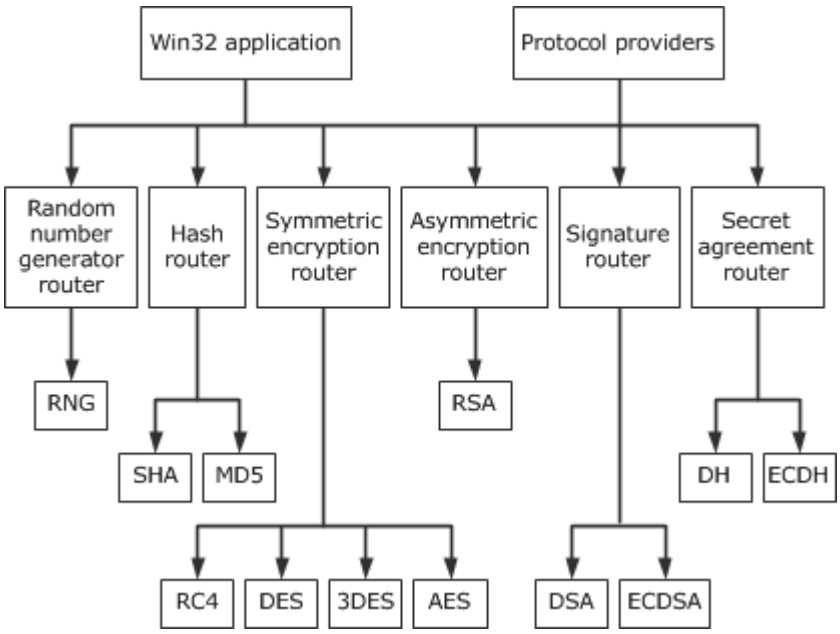


Рисунок 1.1 – CNG криптографічні примітиви

Алгоритми шифрування. Відомо, що в більшості розвинутих країн діють стандарти на алгоритми захисту інформації. Як і в багатьох інших областях, ці стандарти з часом застарівають і підлягають заміні.

Криптоалгоритми діляться (рис. 1.2) на такі категорії [19]:

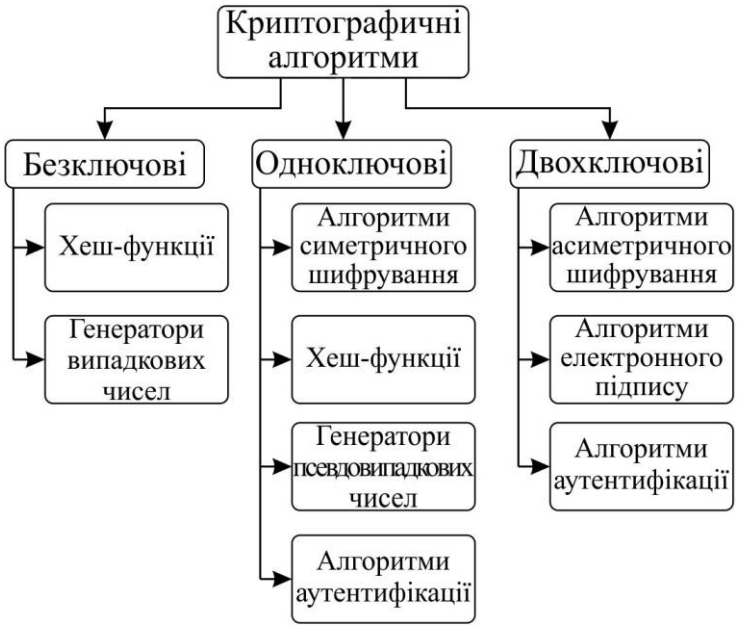


Рисунок 1.2 – Класифікація криптографічних алгоритмів

Безключові – які не використовують яких-небудь ключів в процесі криптографічних перетворень.

Одоключові – використовують в своїх розрахунках деякий секретний ключ.

Двоключові – в яких на різних етапах обчислень використовуються два види ключів: таємній і відкритий.

В рамках даної роботи розглядаютья алгоритми симетричного шифрування. Це такі алгоритми, в яких для шифрування і розшифрування використовується один і той же ключ або ключ розшифрування легко розрахувати з ключа зишифрування. Сама загальна категорія криптоалгоритмів, поділяється на алгоритми блокового (оброблюють інформацію, розділену на блоки) і поточного (шифрують дані побітово) шифрування.

В більшості розвинутих країн використання різних засобів захисту інформації регулюється державою, а алгоритми захисту стандартизовані.

Існує де-факто міжнародний стандарт шифрування AES, в той же час до недавнього часу вітчизняним стандартом шифрування в Україні був ДСТУ ГОСТ 28147:2009 (який замінив розроблений ще в СССР ГОСТ 28147-89). З 1 липня 2015 року в Україні введено в дію новий стандарт шифрування ДСТУ 7624:2014 «Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення» [20, 21].

Ітеративні блочні шифри і їх структура. *Блочна шифросистема* [22] – це симетрична система шифрування з функцією зашифрування, що реалізується базовим блочним алгоритмом зашифрування в конкретному режимі шифрування. Основними режимами є електронна кодова книга (ECB), зчеплення блоків шифротексту (CBC), зворотній зв’язок за шифротекстом (CFB) і зворотний зв’язок з виходом (OFB). В [23] рекомендується використання або режиму лічильника (CTR), або режим *зчеплення шифрованих блоків*.

В подальшому, блочний базовий алгоритм зашифрування будемо називати для стислості *блочним шифром*. Блочний шифр – це алгоритм, що реалізує при кожному фіксованому значенні ключа одне зворотне

відображення множини блоків відкритого тексту на множину блоків шифрованого тексту, що мають фіксований розмір. Основний тип блочних шифрів – це *ітеративні* блочні шифри, що складаються з послідовних однотипних раундів (циклів, ітерацій) шифрування. Для сучасних блочних шифрів розмір блока частіше всього дорівнює 64 або 128 біт. Ключ також являє собою бітову послідовність зазвичай довжиною 128, 192 або 256, з якої згідно процедурі розгортання ключа вираховуються *раундові ключі*.

Для зручності реалізації блочні шифри розроблюються так, щоб функція зашифрування була інволютивною, тобто відрізнялася від розшифрування тільки послідовністю використовуваних раундових ключів. Такий підхід особливо зручний при апаратній реалізації, оскільки достатньо однієї мікросхеми, яка забезпечує і зашифрування, і розшифрування [24].

Іншим класом симетричних шифросистем є поточні шифри, за основу яких обрані генератори послідовностей псевдовипадкових чисел, що залежать від секретного ключа. Головною перевагою поточних шифрів вважається висока швидкодія при шифруванні великих об’ємів даних [24, 25], табл. 1.3.

Таблиця 1.3

Поточні шифри

Назва шифру	Швидкодія, довге повідомлення (до 4000 байт)	Швидкодія, коротке повідомлення (до 40 байт)
AES (режим CTR)	1	1
Hc-128	4,5	0,03
Rabbit	4,1	1,00
Salsa20/12	2,8	0,03
Sosemanuk	2,1	1,02
RC4	1,2	0,06

Принципи побудови ітеративних блочних шифрів. Під час розробки блочних шифрів враховуються строго не формалізовані властивості перемішування, розсіювання і ускладнення, сформульовані К. Шенноном [26]. *Властивість перемішування* [22] означає суттєве ускладнення зв’язку шифротекста і відкритого тексту; *властивість розсіювання* полягає в тому, що

кожний знак відкритого тексту повинен впливати на велику кількість знаків шифротекста; *властивість ускладнення* означає складну залежність між ключом і шифротекстом.

Як один з варіантів що задовольняє перелічені вимоги Шеннон запропонував використовувати композицію транспозицій, лінійних операцій і підстановок [26]. Ці операції лежать в основі багатьох сучасних блочних шифрів. Розглянемо сучасні конструкції блочних шифрів.

SP-сітка і її узагальнення. Концепція *підстановочно-перестановочної сітки* або *SP-сітки* запропонована Х. Фейстелем при описі першого варіанту шифру Lucifer [23]. Цей шифр складається з композиції бітових підстановок, що чергуються, і так званих *S-блоків*, вибір яких залежить від ключа. Перестановки потрібні для забезпечення розсіювання, а нелінійні S-блоки – перемішування і ускладнення. Багато шифрів забезпечують властивість ускладення завдяки складанню блоків тексту і раундового ключа до перетворення, що створюється S-блоками (див. рис. 1.1).

Узагальненнями SP-сітки є XSL-сітка і SA-сітка [27], в яких замість перестановки використовуються відповідно довільні лінійні і афінні перетворення (див. рис. 1.3). Стосовно SA-сітки передбачається, що S-блоки можуть залежати від ключа довільним чином [27]. До XSL-сіток відносяться, наприклад, шифри Rijndael [28] і Serpent [23].

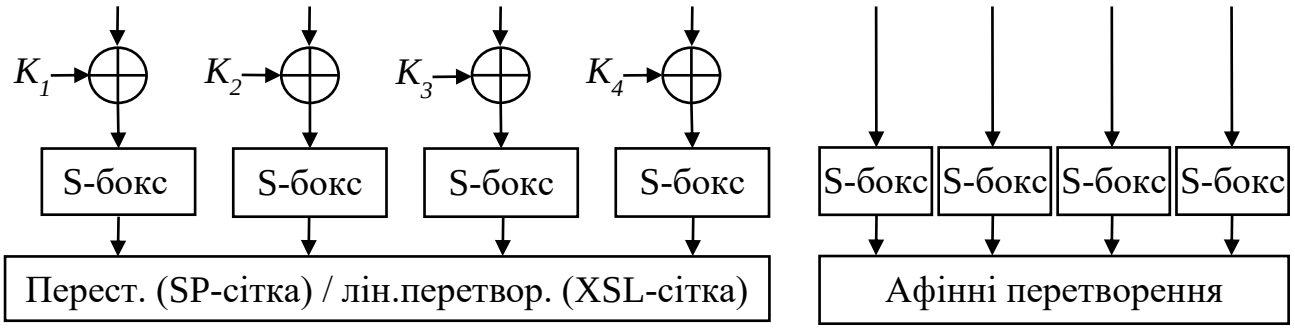


Рисунок 1.3 – SP-сітка і XSL-сітка (зліва) і SA-сітка

Схема Фейстеля. Першими шифрами, де використовувалась сітка Фейстеля були шифри DES [29] і друга версія шифру Lucifer [30]. Раунд такого шифру перетворює блок тексту згідно з наступним алгоритмом (рис. 1.2): розділити вхідний блок на два підблоки; один підблок перетворювати функцією $F : GF\left(2^{s/2}\right) \times GF\left(2^{m}\right) \to GF\left(2^{s/2}\right)$, де s - розмір блоку, а m - розмір ключа раунда; скласти підблоки і поміняти їх місцями.

В роботі [31] виділено підклас схеми Фейстеля, що називається «шифр, схожий на DES» (a DES-like cipher), який характеризується тим, що в ньому залежність функції F від раундового ключа і вхідного блоку має вигляд

$$F(P, K_r) = f(E(P) + K_r), \text{ де } f : GF\left(2^{m}\right) \to GF\left(2^{s/2}\right),\ m \geq s/2,$$

$$E : GF\left(2^{s/2}\right) \to GF\left(2^{m}\right) \text{ – афінне перетворення.}$$

За схемою Фейстеля створений ГОСТ-28147-89 [32], Blowfish [33], 3DES [23].

Існують модифікації сітки ФЕйстеля, що передбачають ділення блока не на два, а на більшу кількість частин (рис. 1.4). Згідно з подібними схемами розроблені, наприклад, шифри MARS [34] і CAST-256 [35].

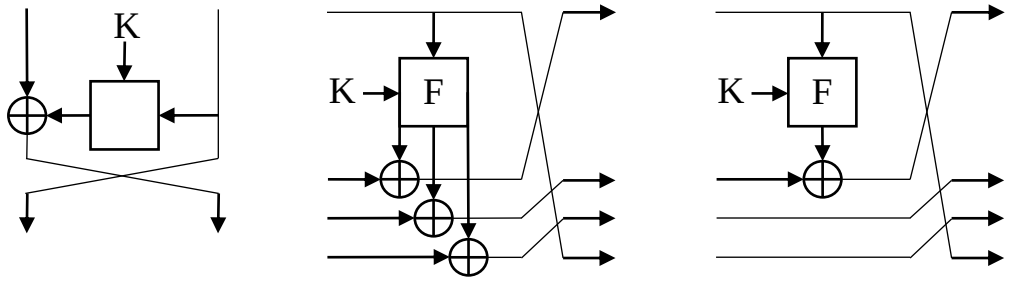


Рисунок 1.4 – Схема Фейстеля (зліва) і її модифікації

ARX-схеми. Ціла низка сучасних блочних шифрів використовують операції, які ефективно реалізуються на ЕВМ. Такими операціями є складання по модулю, циклічні зсуви, операція XOR та інші. До шифрів такого типу відносяться TEA, RC5, HIGHT, HPC, Threefesh [36].

Чергування операцій з різних алгебраїчних груп. Сформульовані Шенноном принципи побудови блочних шифрів в деяких алгоритмах забезпечуються завдяки композиції операцій, що відносяться до різних алгебраїчних груп, наприклад операція XOR, складання по модулю 2^{16}, множення по модулю $2^{16}+1$, а також перестановкою підблоків за аналогією зі схемою Фейстеля і її модифікаціями. Шифрами цього типу є PES, IDEA, MESH [36].

Відбілювання. Деякі сучасні шифри використовують компонент, який має назву *відбілювання* (whitening); ця операція являє собою додавання (віднімання, операцію XOR) з ключем «відбілювання». Шифри, що використовують дану операцію, MARS, RC6, HIGHT. Відзначається, що операція відбілювання практично не впливає на продуктивність, але дозволяє нейтралізувати ряд потенційних загроз [37].

Стандартизація і застосування блочних шифрів. Ітеративні блочні шифри широко представлені в комерційних і державних стандартах, а також в бібліотеках забезпечення інформаційної безпеки сучасних мов програмування.

Єдиний міжнародний стандарт шифрування даних в країнах СНГ визначено ГОСТом 28147-89 [38], де описано 64-бітний блочний шифр з 256-бітним ключом. Даний шифр використовується для вироблення коду автентичності повідомлення, на його базі побудований старий стандарт на хеш-функцію, який визначається ГОСТом Р34.11-94 [39]. Зараз використовується ГОСТ Р34.11-2012.

Блочні шифри AES [40] і DES [41] це відповідно нові і старі стандарти США. Фіналістами конкурсу AES стали Rijndael, Twofish, MARS, RC6 і Serpent, які показали ефективну роботу на різних платформах і високу стійкість [20].

В рамках проекту NESSIE і CRYTPREC досліджувались блочні шифри та інші методи захисту інформації [42].

Crypto++ - це бібліотека з відкритим вихідним кодом на C++, яка містить велику кількість методів і засобів захисту інформації, серед яких асиметричні шифросистеми, поточні і блочні шифри, коди автентичності повідомлень, хеш-функції, шифросистеми на еліптичних кривих. В набор блочних шифрів включені кандидати конкурсу AES Rijndael, RC6, Twofish, MARS, Serpent і CAST-256, а також IDEA, 3DES, Camellia, RC5, Blowfish, TEA, XTEA, Skipjack. Деякі блочні шифри, реалізовані з цією бібліотекою, класифіковані, як «нестійкі або застарілі», зокрема, DESX, RC2, Safer, Shark [43].

Платформа .Net і мова C# надає програмістам засоби захисту інформації завдяки бібліотеці System.Security.Cryptography, яка разом з іншими засобами захисту інформації містить блочні шифри DES, 3DES, AES і RC2. Класи, призначені для забезпечення інформаційної безпеки в мові Java, розташовані в пакеті java.security, що містить блочні шифри AES, 3DES і RC2. Для доступу до методів захисту інформації на мові PHP слід використовувати бібліотеку MCrypt [122], в якій представлені наступні блочні шифри: Blowfish, DES, 3DES, Twofish, TEA, RC2 [44].

1.2. Класифікація та кратка характеристика алгоритмів криптографічного захисту інформації

В даному параграфі наведена класифікація та надана коротка характеристика алгоритмів криптографічного захисту інформації, що включає в себе опис формування міжнародних стандартів шифрування, ГОСТ 28147, Rijndael (AES), ДСТУ 7624:2014 («Калина»).

Формування міжнародних стандартів шифрування. Алгоритм DES (Data Encryption Standard), розроблений корпорацією IBM з 1977 був федеральним стандартом шифрування даних США. Використовувався не тільки урядом США, але і отримав широке розповсюдження по всьому світу серед приватних користувачів. Зі зростанням розрахункової потужності комп'ютерів почали виникати питання щодо криптосійкості DES'а методом взлому «груба сила», але стандарт проходив повторні сертифікації, що проводились в 1983,

1988 і 1993. В середині 90х років стало очевидна невідповідність загальноприйнятого стандарту шифрування DES сучасним вимогам. В першу чергу із-за недостатньої довжини ключа всього 56 біт. За даними Брюса Шнайера вже в 1993 році існували пристрої здатні взламати DES за прийнятний час.

Процес розробки нового федерального інформаційного стандарту (FIPS) для шифрування даних Advanced Encryption Standard (AES) був ініційований Національним Інститутом стандартів і технологій (NIST).

На початку січня 1997 року NIST повідомило про початок розробки AES, опублікувавши документ “Announcing development of a federal information processing standard for advanced encryption standard”, що містив первинні вимогу до алгоритму [22]:

1) Алгоритм шифрування AES повинен бути відкрито опублікований;
2) Алгоритм повинен бути симетричним блочним шифром;
3) AES повинен передбачати можливість збільшення довжини ключа;
4) AES повинен бути легко реалізуємий як апаратно так і програмно;
5) AES повинен поширюватись безкоштовно і бути загальнодоступним і рамках патентів ANSI.

У вересні 1997 року вийшов уточнюючий документ “Call for AES Candidate Algorithms”, оголошуючий о проведенні конкурсу на AES, він містив офіційні вимоги до кандидатів. Зокрема алгоритм повинен підтримувати наступні комбінації довжин блоків і ключа 128-128, 128-192 і 129-256 бітів. До 15 червня 1998 року було заявлено 21 криптографічний алгоритм (табл. 1.4), але тільки 15 з них задовольняли початкові вимоги.

Таблиця 1.4.

Криптографічні алгоритми, що сприяли формуванню міжнародних стандартів шифрування

Країна походження	Алгоритм	Автори
Австралія	LOKI97	Lawrie Brown, Josef Pieprzyk, Jennifer Seberry
Бельгія	RIJNDAEL	Joan Daemen, Vincent Rijmen
Канада	CAST-256	Entrust Technologies, Inc

Продовження табл. 1.4

	DEAL	Richard Outerbridge, Lars Knudsen
Коста-Ріка	FROG	TecApro Internacional S.A.
Франція	DFC	Centre National pour la Recherche Scientifique
Німеччина	MAGENTA	Deutsche Telekom AG
Японія	E2	Nippon Telegraph and Telephone Corporation
Корея	CRYPTON	Future Systems, Inc
США	HPC	Rich Schroeppel
	MARS	IBM
	RC6	RSA Laboratories
	SAFER+	Cylink Corporation
	TWOFISH	Bruce Schneier, John Kelsey, Doug Whiting, David Wagner, Chris Hall, Niels Ferguson
Великобританія, Ізраїль, Норвегія	SERPENT	Ross Anderson, Eli Biham, Lars Knudsen

В кінці серпня 1998 року в Каліфорнії відбулась конференція, присвячена відбору кандидатів на AES, отримавша назву AESI.

Серед основних критеріїв оцінювання алгоритмів були [24]:

1) **Криптостійкість.** Проводилось порівняння алгоритмів на ступінь незалежності вихідного блоку від випадкової перестановки бітів вхідного блоку, схильність до відомих криптоатак. Кожен кандидат повинен був надати оціночну криптосійкість алгоритму;
2) **Вартість.** Кандидат надавав інформацію о ліцензійних уходах і патентах на алгоритм. Також враховувалась продуктивність алгоритму (швидкість шифрування), необхідний для шифрування розмір пам’яті.
3) **Особливості алгоритму** і його реалізація. Гнучкість, апаратна і програмна зручність реалізації.

В квітні 1999 року в Римі відбулась конференція AES2, в межах якої відбувались порівняння продуктивності програмних засобів реалізації алгоритмів з оптимізацією під мови програмування C і Java. Найбільшу швидкість зашифрування/розшифрування показав алгоритм Crypton (40Мбайт/с), найменшу Magenta і HPC (2Мбайта/с). Хоча при проведенні тестів

на різних платформах і з різними компіляторами, отримані результати досить сильно відрізнялись.

Усі блочні алгоритми можна поділити на дві основні групи:

1) Ті що використовують мережу (сітку) Фейстеля;
2) Ті що використовують мережу перестановок-підстановок (SP-мережі) що базуються на послідовних перестановках і підстановках, залежних від ключа.

Серед алгоритмів-претендентів до першої групи відносяться CAST-256, DEAL, DFC, E2, LOKI97, MAGENTA, MARS, RC6, TWOFISH, до другої групи CRYPTON, Rijndael, SAFER+ и SERPENT. Алгоритми FROG і HPC не підпали під жодну з цих категорій, але в ході обговорення кандидатів не виявлено яких небудь видатних якостей цих алгоритмів.

9 серпня NIST надрукувала прес-реліз “Announces AES Finalists”, в якому оголошувались п’ять фіналістів: MARS, RC6, Rijndael, Serpent, TwoFish.

Третя конференція AES пройшла в Нью-Йорку 13 і 14 квітня 2000 року, незадовго до завершення другого етапу конкурсу. На якій був проаналізований Rijndael зі скороченою кількістю раундів і показана його слабкість в такому випадку. 2 жовтня 2000 року було оголошено, що переможець конкурсу стає алгоритм Rijndael, і почалась процедура стандартизації. 28 лютого 2001 року був опублікований проект, а 26 листопада 2001 року AES був прийнятий як FIPS 197 [40]. Алгоритм Rijndael став переможцем конкурсу AES і став новим стандартом США і де-факто – стандартом симетричного шифрування майже в усьому світі.

Ще до закінчення конкурсу AES, в січні 2000 року почався досить схожий конкурс в Європі, що передбачав обрання криптостандартів Євросоюзу, – конкурс NESSIE (New European Schemes for Signature, Integrity and Encryption – «нові європейські алгоритми електронного підпису, забезпечення цілісності і шифрування»). Організатори конкурсу NESSIE ставили перед собою значно більші цілі, ніж американський інститут NIST. На конкурсі навіть знайшлось місце для криптоаналітичного алгоритму, тобто алгоритму пошуку слабких

місць в алгоритмах шифрування. Крім алгоритмів надісланих на конкурс, розглядались і відомі криптографічні стандарти і алгоритми: ті ж учасники конкурсу AES – алгоритми RC6 і Rijndael в категорії алгоритмів блочного шифрування.

Результатом вивчення алгоритмів став звіт експертів-криптографів “NESSIE security report”, опублікований на офіційному сайті конкурсу [38]. Не дивлячись на те, що конкурс NESSIE закінчився у лютому 2003 року загальноєвропейських криптографічних стандартів ще досі нема.

Алгоритм шифрування ГОСТ 28147-89 визначено в стандарті [6], Алгоритм (рис. 1.5) шифрує дані 64-бітними блоками з використанням 256-бітного ключа шифрування. Виконується 32 раунда перетворень, в кожному з яких передбачені наступні операції:

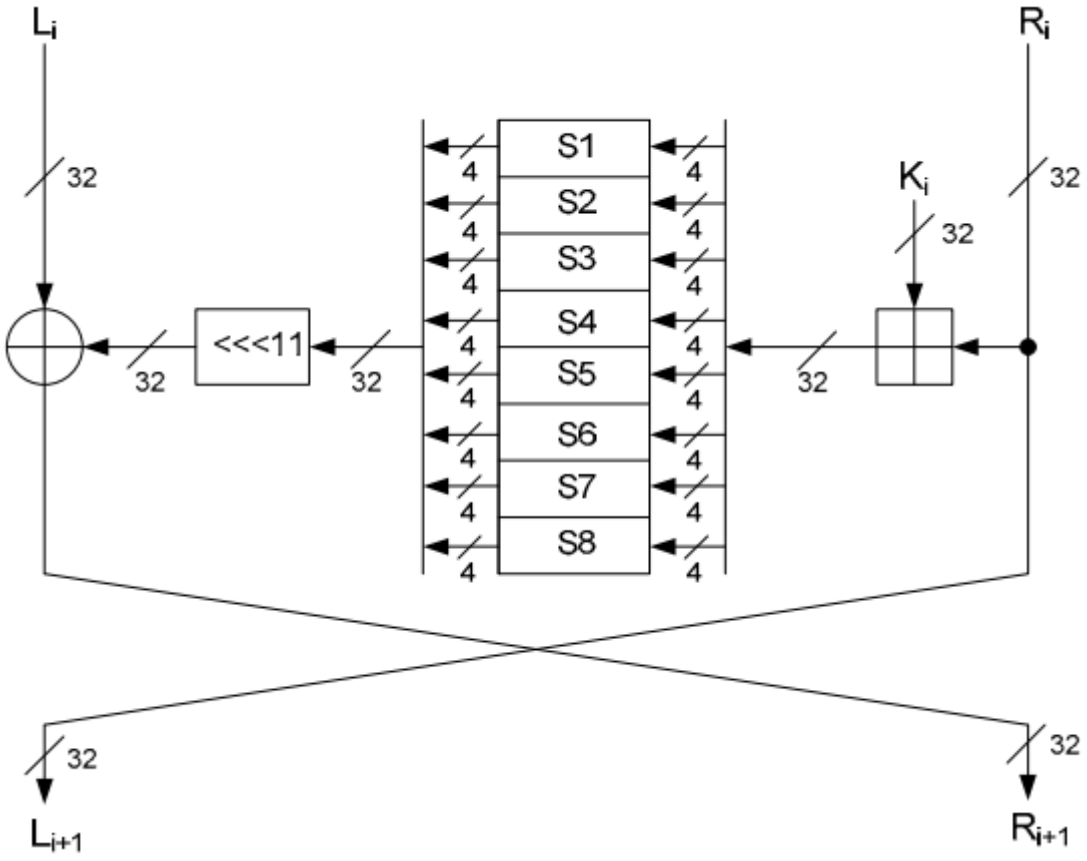


Рисунок 1.5 – Структурна схема раунду алгоритму ГОСТ 28147

Перед початком шифрування 64-бітний блок відкритого тексту заноситься в 32-бітні регістри $L_0 \parallel R_0$.

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

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

де:

$\oplus$ – позначає побітову суму за модулем 2;

$<<<11$ – циклічний зсув вліво на 11 розрядів.

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

Процедура шифрування за ГОСТ 28147.

1) Один з 32-бітних субблоків даних складається з 32-бітним значенням ключа раунда K_i по модулю 232;

2) Результат попередньої операції поділяється на 8 фрагментів по 4 біта, які паралельно «проходять» через 8 таблиць замін $S_1 \ldots S_8$ (вузли заміни). Таблиці замін в стандарті не визначені. Приклади можливих таблиць можна знайти [45];

3) 4-бітні фрагменти (після заміни) об’єднуються знову в 32-бітний субблок, значення якого циклічно зсувається вліво на 11 біт;

4) Оброблений попередніми операціями субблок накладається на необроблений на необроблений субблок за допомогою побітової логічної операції «виключаюче або» (XOR);

5) Субблоки міняються місцями.

Процедура розширення ключа в алгоритмі ГОСТ 28147-89, фактично, відсутня: в раундах шифрування послідовно використовується 32-бітні фрагменти $K_1 \dots K_8$ початкового 256-бітного ключа шифрування в наступному порядку: K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, – за виключенням останніх 8 раундів – в раундах з 25-го по 31-й фрагменти використовуються в зворотному порядку.

В алгоритмі ГОСТ 28147 не використовується процедура розширення ключа. Ключ K довжиною 256 біт розглядається як вісім 32-бітних підключів: $K = K_0 \parallel K_1 \parallel K_2 \parallel K_3 \parallel K_4 \parallel K_5 \parallel K_6 \parallel K_7$. У раундах шифрування з номерами $0 \le r \le 23$ раундовий підключ K_i визначається як $K_i = K_{(r \bmod 8)}$, а для останніх восьми раундів $24 \le r \le 31$ як $K_i = K_{7-(r \bmod 8)}$. Під час розшифрування порядок підключів зворотній.

Розшифрування повністю аналогічно зашифруванню, але з іншим порядком використання фрагментів ключа:

1) В прямому порядку – в перших 8 раундах;
2) В інших раундах – в зворотньому порядку.

Стандарт [38] також передбачає і опис різних режимів використання алгоритму:

1) Описаний вище режим простої заміни;
2) Режим гамування і гамування зі зворотним зв’язком, що передбачає розрахунки за допомогою описаних вище перетворень псевдовипадкової послідовності – гама шифру – і її накладання на шифруємий текст.
3) Режим обчислення імітовставки – криптографічної контрольної суми, що використовується для підтвердження цільності даних; в цьому режимі виконується 16 раундів перетворень замість 32-х.

На базі ГОСТ 28147-89 побудовано стандарт хешування ГОСТ Р 34.11-94.

Розвиток ГОСТ 28147-89. В 2011 р. ГОСТ 28147-89 подавався на голосування, але не був включений в міжнародний стандарт блочного шифрування ISO/IEC 18033-3.

Стосовно практичної реалізації, то у ГОСТ 28147-89 основні операції виконуються над 32-бітними «напівблоками», що зумовлює високу швидкість роботи алгоритму на 32-бітних платформах. Порівняно з байт-орієнтовним алгоритмом AES на 8-бітних платформах ГОСТ програє в швидкості в 4 рази [18].

ГОСТ 28147-89 має декілька модифікацій, які визнані не кращими за оригінал.

AES – 32 операції в раунді і 14 раундів, це 448 операцій.

ГОСТ – 10 операцій в раунді і 32 раунді, це 320 операцій.

(величини одного порядку).

Алгоритм Rijndael (шифр AES) [20, 40]. Для шифрування (рис. 1.6) Rijndael використовує 4 кроки (байтова нелінійна заміна, рядковий зсув, перемішування стовбців і додавання ключа) від 10 до 14 повторюваних раундів. Це було запозичено з попереднього шифру Square, також розробленого Daemen і Rijmen.

AES має фіксований розмір блоку 128 біт і розмір ключа 128, 192 або 256 біт, в той час як Rijndael може бути зі змінним розміром блоку і ключа. Розмірами в будь-якому кратному 32 біт значеннями, з мінімум 128 біт і максимум 256 біти.

Тому актуальні зараз AES-128, AES-192 і AES-256.

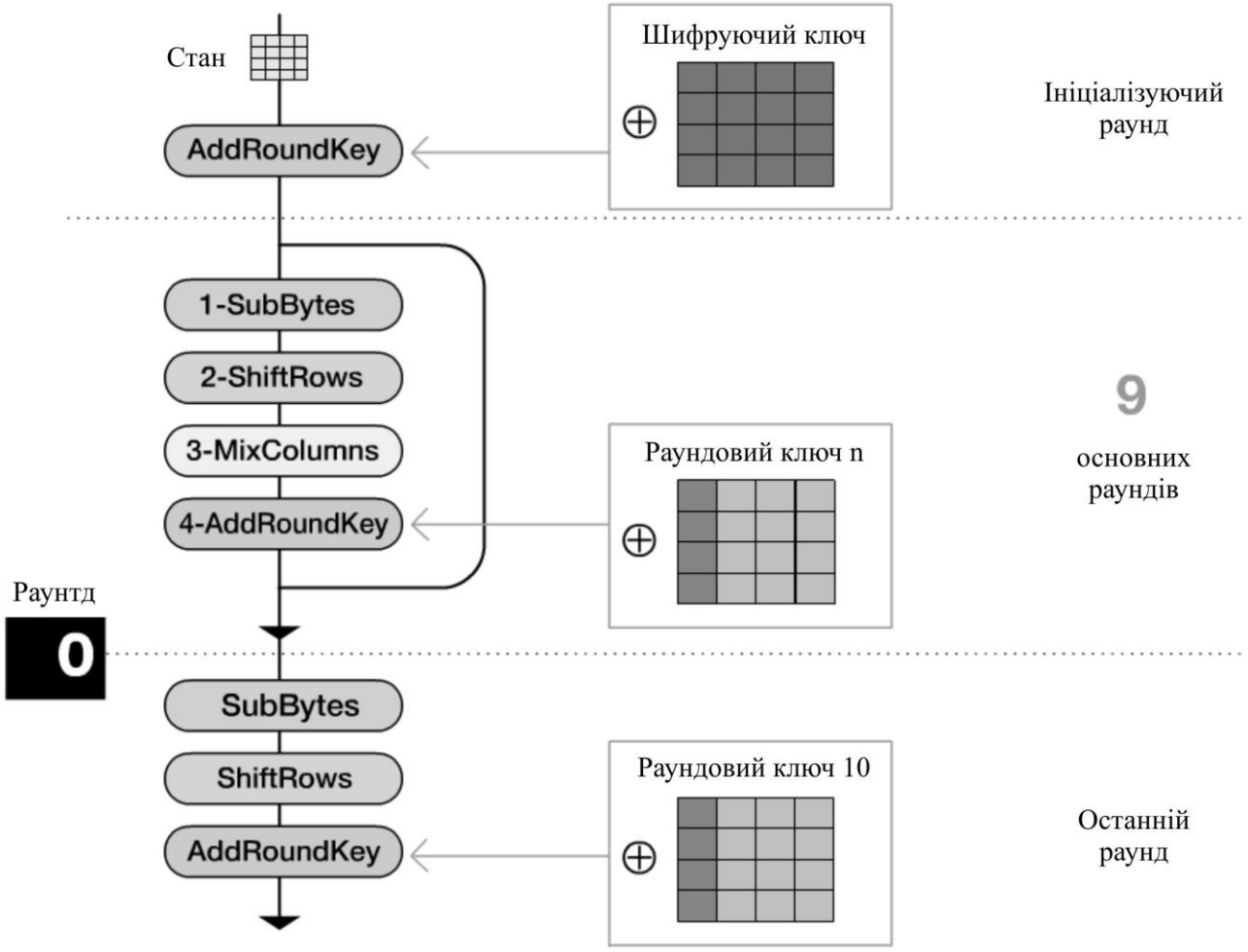


Рисунок 1.6 – Процес шифрування Rijndael

Блок-схема алгоритму наведена на рис. 1.7, а далі дається короткий опис функціональних перетворень

1) ExpandKey – функція для розрахунку усіх раундових ключів;
2) SubBytes – функція для підстановки байт, використовуючи таблицю підстановок (нелінійна підстановка) S-Box;
3) ShiftRows – функція, що забезпечує циклічний зсув в формі на різні величини;
4) MixColumns – функція, яка змішує дані всередині кожного стовбця форми;
5) AddRoundKey – складання ключа раунда з формою.

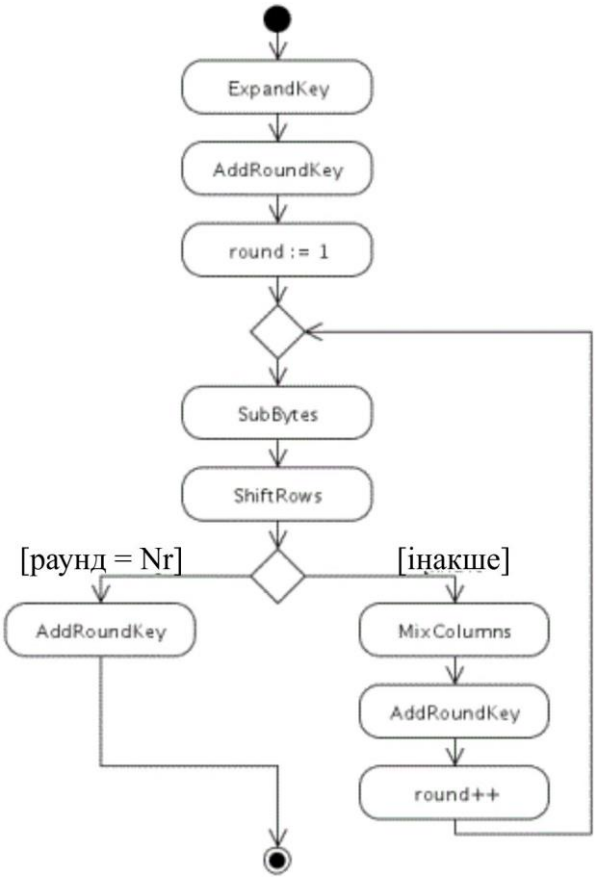


Рисунок 1.7 – Блок-схема шифратора Rijndael

S-Box (таблиця підстановок) – таблиця, яка задає відображення (табл. 1.5) одного байту в інший (бієктивне відображення).

Таблиця 1.5

Таблиця підстановок для шифратора Rijndael

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

Наприклад, {fe} замінюється на {bb}.

Зворотній S-Box (зворотня таблиця підстановок) – аналогічно S-Box, задає (табл. 1.6) зворотнє відображення.

Таблиця 1.6

Зворотня таблиця підстановок для шифратора Rijndael

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

Розшифрування відбувається так само, тільки в зворотному порядку. Повний опис можна прочитати в NIST Fips-197 [40].

ДСТУ 7624:2014 (“Калина”) [21].

Впроваджений в липні 2015 року в якості українського стандарту шифр Калина підтримує кілька варіантів довжин блоку і ключа.

Внутрішній стан шифру є матрицею з 8 рядами і 2 стовпцями. Перед початком шифрування матриця заповнюється байтами відкритого тексту. Потім над елементами матриці виробляється 10 раундів наступних перетворень (рис. 1.8).

$\eta_i^{(K_v)}$ – Складання колонок матриці стану з колонками раундового ключем, представленого у вигляді матриці, по модулю 2^{64}.

$\pi_l^{'}$ (SubBytes) – заміна кожного байта матриці стану підстановкою з однією з чотирьох таблиць π_0, π_1, π_2, π_3.

τ_l (SiftRows) – циклічний зсув вправо на одну позицію рядків з 4-го по 8-й.

ψ_l (MixColumns) – перетворення колонок матриці станів. Кожен елемент нової матриці обчислюється за формулою:

$$\omega_{i,j} = (\upsilon >>> i) \otimes G_j,$$

де $\otimes$ – скалярний добуток векторів, υ – вектор, $\upsilon = (0x01, 0x01, 0x05, 0x01, 0x08, 0x06, 0x07, 0x04)$,

G_j – колонка матриці. Операції множення і складання виробляється в кінцевому полі по модулю многочлена $\mathrm{T} = x^8 + x^4 + x^3 + x^2 + 1$

$k_l^{(K_\upsilon)}$ – Побітове додавання по модулю 2 матриці внутрішнього стану шифру і раундового ключа K_υ

Процес шифрування описується наступним виразом:

$$T_{l,k}^{(K)} = \eta_l^{(K_l)} \circ \psi_l \circ \tau_l \circ \pi_l^{'} \circ \prod_{\nu=1}^{t-1} \left(k_l^{(K_\nu)} \circ \psi_l \circ \tau_i \circ \pi_l^{'} \right) \circ \eta_l^{(K_0)}$$

Для формування раундовий ключів спочатку, використовуючи майстер-ключ K, обчислюється проміжний ключ K_σ

$$\Theta^{(K)} = \psi_l \circ \tau_l \circ \pi_l^{'} \circ \eta_l^{(K_\alpha)} \circ \psi_l \circ \tau_l \circ \pi_l^{'} \circ k_l^{(K_\omega)} \circ \psi_l \circ \tau_l \circ \pi_l^{'} \circ \eta_l^{(K_\alpha)},$$

де $K_\alpha = K_\omega = K$ для випадків, коли довжина блоку дорівнює довжині ключа.

Ключі для парних раундів генеруються на основі проміжного ключа: $\Xi^{(K,K_\sigma,i)} = \eta_l^{\left(\varphi_i^{(K_\sigma)}\right)} \circ \psi_l \circ \tau_l \circ \pi_l^{'} \circ k_l^{\left(\varphi_i^{(K_\sigma)}\right)} \circ \psi_l \circ \tau_l \circ \pi_l^{'} \circ \eta_l^{\left(\varphi_i^{(K_\sigma)}\right)}$, де i – номер раунду, а $\varphi_i^{(K_\sigma)} = \eta_l^{(K_\sigma)} \left(\upsilon << \left(\frac{i}{2} \right) \right)$ і $\upsilon = \mu_l^{0x00010001 \ldots 0001}$

Або в більш наочному вигляді:

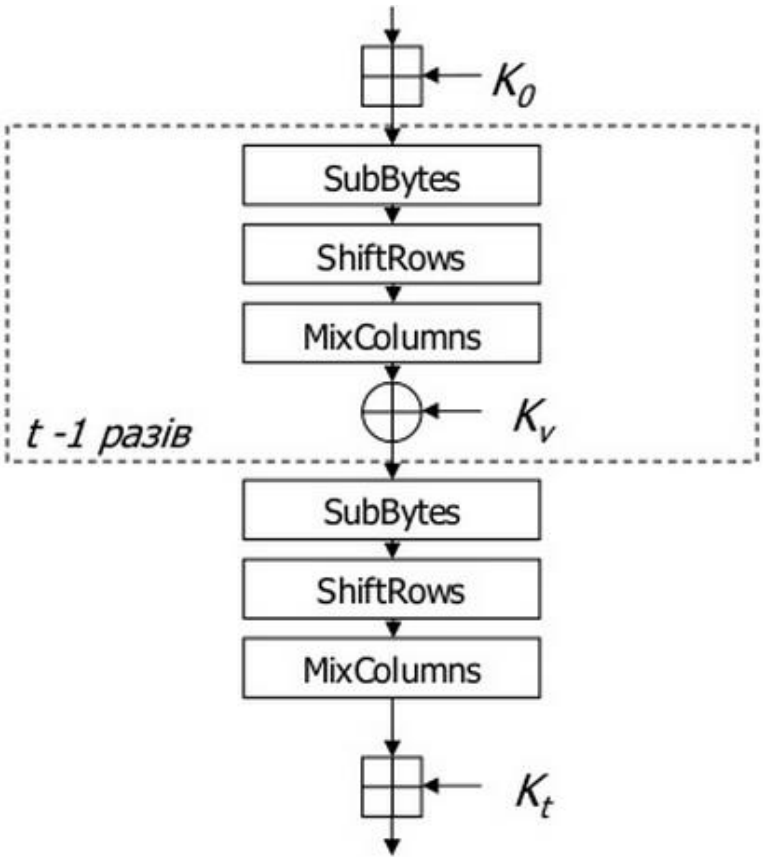


Рисунок 1.8 – «Калина»: функція зашифрування

Ключі для непарних раундів обчислюються так: $K_i = \left(K_{i-1} <<< \left(\frac{l}{4} + 24 \right) \right)$,

де l – довжина блоку.

При розшифрування використовуються зворотні перетворення:

$$U_{l,k}^{(K)} = -1\eta_l^{(K_0)} \circ \prod_{\upsilon=t-1}^{1} \left(-1\pi_l^{'} \circ -1\tau_l \circ -1\psi_i \circ k_i^{(K_\upsilon)} \right) \circ -1\pi_l^{'} \circ -1\tau_l \circ -1\psi_i \circ \eta_i^{(K_l)}$$

ГОСТ Р 34.12-2015 («Кузнечик») [46].

Стандарт вступив в дію з 1 січня 2016 року. Цей стандарт крім старого ГОСТ 28147-89, який тепер називається «Магма» і має фіксований набір підстановок, містить опис блочного шифру «Кузнечик».

На відміну від ГОСТ 28147-89 новий шифр являє собою не мережу Фейстеля, а т.зв. SP-мережу: перетворення, що складається з декількох

однакових раундів, при цьому кожен раунд складається з нелінійного та лінійного перетворень, а також операції накладення ключа. На відміну від мережі Фейстеля, при використанні SP-мережі перетворюється весь вхідний блок, а не його половина. Така структура іноді також називається AES-like (схожою на AES), проте, на відміну від останнього у «Кузнечика» є ряд своїх відмінностей:

- лінійне перетворення може бути реалізовано в за допомогою регістра зсуву;
- ключова розгортка реалізована за допомогою мережі Фейстеля, в якій в якості функції використовується раундове перетворення вихідного алгоритму.

Довжина вхідного блоку «Кузнечика» - 128 біт, ключа - 256 біт.

Шифрування засноване на послідовному застосуванні декількох однотипних раундів, кожен з яких містить три перетворення: додавання з раундовим ключем, перетворення блоком підстановок і лінійне перетворення (рис.1.9).

128-бітний вхідний вектор чергового раунду складається побитно з раундовим ключем:

$X[k]: V_{128} \to V_{128}$ $\qquad X[k](a) = k \oplus a$, де $k, a \in V_{128}$;

Нелінійне перетворення являє собою застосування до кожного 8-бітного підвектора 128-бітного вхідного вектора фіксованою підстановки:

$S: V_{128} \to V_{128}$ $\qquad S(a) = S(a_{15} \| \ldots \| a_0) = \pi(a_{15}) \| \ldots \| \pi(a_0),$

де $a = a_{15} \| \ldots \| a_0 \in V_{128}, a_i \in V_8, i = 0,1,\ldots,15;$

У «Кузнечик» використовується та ж підстановка, що і в хеш-функції «Стрибог».

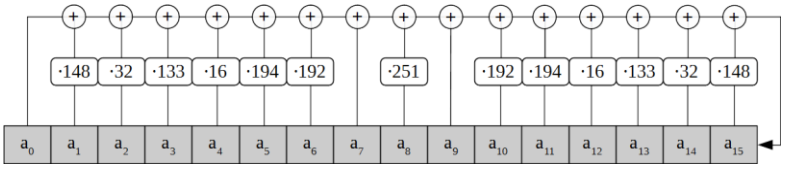
Лінійне перетворення може бути реалізовано не тільки як зазвичай в блокових шифри - матрицею, але і за допомогою лінійного регістра зсуву зі зворотним зв'язком, який рухається 16 разів.

$R: V_{128} \to V_{128}$ $\quad R(a) = R(a_{15} \| \ldots \| a_0) = l(a_{15}, \ldots, a_0) \| a_{15} \| \ldots \| a_1,$

де $a = a_{15} \| \ldots \| a_0 \in V_{128}, a_i \in V_8, i = 0,1,\ldots,15;$

$L: V_{128} \to V_{128}$ $\quad L(a) = R^{16}(a)$, де $a \in V_{128}$.

Сам реєстр реалізується над полем Галуа по модулю неприводимого многочлена ступеня 8: $p(x) = x^8 + x^7 + x^6 + x + 1$



Раундові перетворення можна зобразити таким чином:

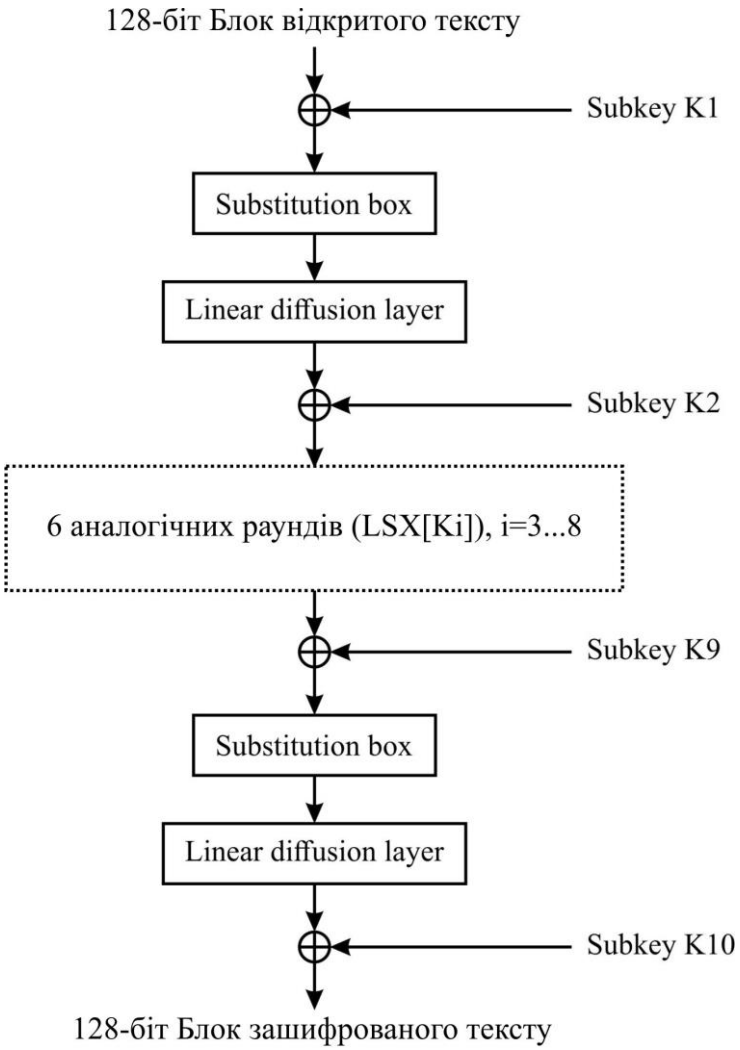


Рисунок 1.9 – Шифр «Кузнечик»

Шифрування одного 128-бітного вхідного блоку описується наступним рівнянням:

$$E_{K_1,\ldots,K_{10}}(a) = X[K_{10}]LSX[K_9]\ldots LSX[K_2]LSX[K_1](a),$$

Розшифрування реалізується зверненням базових перетворень і застосуванням їх в зворотному порядку:

$$D_{K_1,\ldots,K_{10}}(a) = X[K_1]S^{-1}L^{-1}X[K_2]\ldots S^{-1}L^{-1}X[K_9]S^{-1}L^{-1}X[K_{10}](a).$$

СТБ 34.101.31-2007 («BelT») [47].

Має довжину блоку 128 біт і довжину ключа 256 біт, був прийнятий в якості стандарту симетричного шифрування республіки Білорусь у 2011 році. Шифрування здійснюється 8 раундами перетворень, що застосовуються до вхідного блоку.

Процедура шифрування складається з наступних кроків:

- Вхідний блок записується у вигляді

  $X = X_1 \| X_2 \| X_3 \| X_4, X_i$

- Ключ записується у вигляді $\theta = \theta_1 \| \theta_2 \| \theta_3 \| \theta_4 \| \theta_5 \| \theta_6 \| \theta_7 \| \theta_8, \theta_i$ і визначаються раундові ключі

  $K_1 = \theta_1, K_2 = \theta_2, \ldots, K_8 = \theta_8, K_9 = \theta_1, K_{10} = \theta_2, \ldots, K_{56} = \theta_8$

  Більш детально процес зашифрування показан на рис.1.10.

  Для шифрування слова X на ключі θ необхідно виконати наступні кроки:

  1. Встановити у додаткові змінні $a \leftarrow X_1, b \leftarrow X_2, c \leftarrow X_3, d \leftarrow X_4$
  2. Для i = 1,2, ..., 8 виконати:

1) $b \leftarrow b \oplus G_5(a \boxplus K_{7i-6})$;
2) $c \leftarrow c \oplus G_{21}(d \boxplus K_{7i-5})$;
3) $a \leftarrow a \boxminus G_{13}(b \boxplus K_{7i-4})$;
4) $e \leftarrow G_{21}(b \boxplus c \boxplus K_{7i-3}) \oplus \langle i \rangle_{32}$;
5) $b \leftarrow b \boxplus e$;
6) $c \leftarrow c \boxminus e$;
7) $d \leftarrow d \boxplus G_{13}(c \boxplus K_{7i-2})$;
8) $b \leftarrow b \oplus G_{21}(a \boxplus K_{7i-1})$;
9) $c \leftarrow c \oplus G_5(d \boxplus K_{7i})$;
10) $a \leftrightarrow b$;
11) $c \leftrightarrow d$;
12) $b \leftrightarrow c$.

3. Встановити $Y \leftarrow c \| a \| d \| b$
4. Повернути Y.

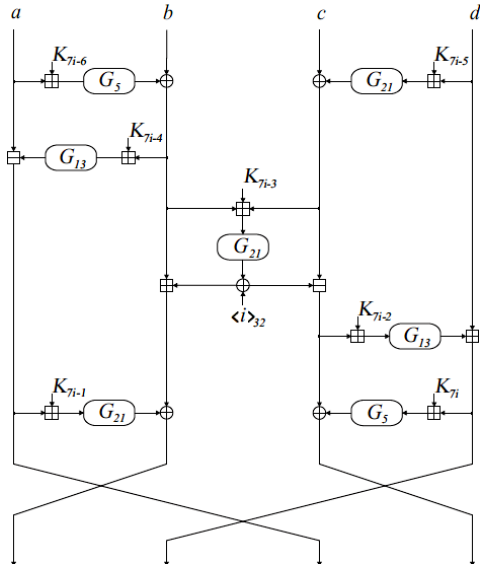


Рисунок 1.10 – Розрахунки на i-м раунді

Змінні a, b, c, d, e - 32-бітові слова. Перетворення $G_{r=(5,13,21)}$ (операція перетворення 32-бітної вхідного рядка) ставить у відповідність до 32-бітного слова $u = u_1 \| u_2 \| u_3 \| u_4$, ($u_i$ – 8-бітне слово), 32-бітне слово.

$$G_r(u) = RotHi^r\left(H(u_1) \| H(u_2) \| H(u_3) \| H(u_4)\right)$$

Підстановка $H : \{0,1\}^8 \rightarrow \{0,1\}^8$ задається фіксованою таблицею (S-блок).

При розшифрування ті ж операції застосовуються в зворотному порядку.

1.3. Основні атаки та критерії ефективності алгоритмів СБШ

Показані основні атаки на шифри та критерії ефективності алгоритмів СБШ. Наведено основні поняття криптоаналізу. Розглянуто такі методи криптоаналізу: Різнисний метод і його різновиди; Лінійний метод; Інтегральний метод; Статистичні методи та ін.

Поняття криптоаналізу.

Криптоаналізом (від грецького krypts – «прихований» і analein – «послаблювати» або «позбавлювати») називають науку відновлення (розшифрування) відкритого тексту без доступу до ключа [48].

Спроба криптоаналізу називається *атакой*. Криптоаналіз ставить своєю метою в різних умовах отримати додаткові відомості про ключ шифрування, щоб значно зменшити діапазон можливих ключів. Результати криптоаналізу можуть варіюватись за ступенем практичного застосування. Криптограф Ларс Кнудсан пропонує наступну класифікацію успішних результатів криптоаналізу блочних шифрів в залежності від об’єму і якості таємної інформації, яку вдалось отримати [49]:

1) Повний злом – криптоаналітик отримує секретний ключ;
2) Глобальна дедукція – криптоаналітик розробляє функціональний еквівалент досліджуємого алгоритму, що дозволяє зашифровувати і розшифровувати інформацію без знання ключа;
3) Часткова дедукція – криптоаналітику вдається розшифрувати або зашифрувати деяке повідомлення;
4) Інформаційна дедукція – криптоаналітик отримує деяку інформацію про відкритий текст або ключ.

Злом шифру не завжди має на увазі виявлення способу, застосованого на практиці для відновлення відкритого тексту маючи перехоплене зашифроване повідомлення. Припустимо, для розшифрування тексту методом повного

перебору потрібно перебрати 2^{128} можливих ключів; тоді винаходження способу, що потребує для розшифрування 2^{110} операцій з підбору ключа, буде вважатись зломом. В науковій криптолоогії [50] під зломом розуміють лише підтвердження наявності вразливості криптоалгоритму, що свідчить, що властивості шрифта не відповідають заявленим характеристикам. Як правило, криптоаналіз починається зі спроби злому спрощеної модифікації алгоритму, після чого результат поширюється на повноцінну версію.

Виявлені апаратні уразливості шифраторів (англ. backdoor або trapdoor) рис.1.11.

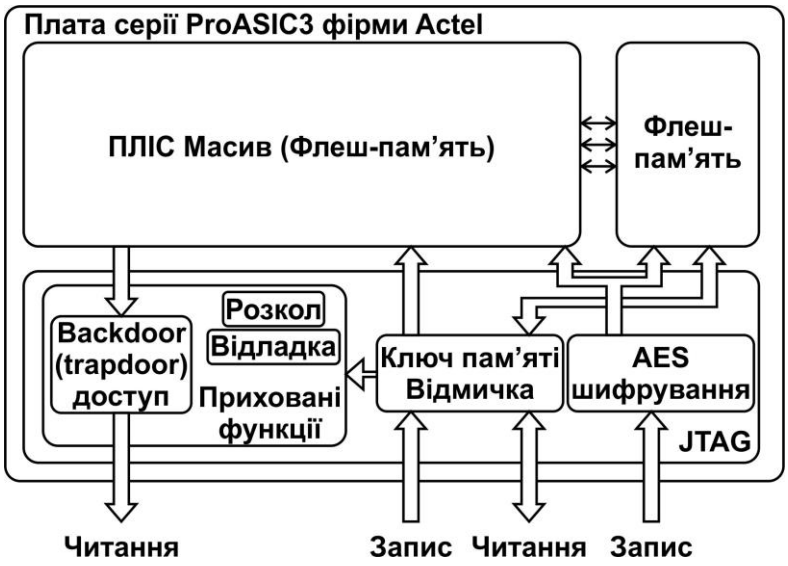


Рисунок 1.11 – Структурна схема плати серії ProASIC3 фірми Actel з апаратною уразливістю шифру AES

Виникнення нових криптографічних алгоритмів призводить к розробці методів їх злому. Якщо метою криптоаналітика є розкриття якомога більшої кількості шифрів, то для нього найкращою стратегією є розробка універсальних методів аналізу [51, 52]. Проте ця задача є також і найбільш складною. Результатом виникнення кожного нового методу криптоаналізу є перегляд оцінок безпечності шифрів, що, в свою чергу, спричиняє необхідність більш

стійких шифрів. Таким чином, історичні етапи розвитку криптографії і криптоаналізу нерозривно зв’язані.

Таблиця 1.8

Порівняльний аналіз сучасних шифрів
на мікроконтролерах з архітектурою AVR і ARM, а також FPGA

Шифр	Довжина ключа, (біт)	Завантаженість МК, (%)	Статистична стійкість	Уразливість (backdoor)	Ресурсо-ємність, (%)
AES	128, 192, 256	10…90	1	Так	10…100
3DES	112, 168	70	0,9	Так	30
RC4	40–1024	10	0,7	Так	50
ДСТУ 7624:2014	128, 256, 512	80	1	Ні	100
ГОСТ Р 34.12-2015	256	80	1	Ні	100

На рис. 1.12 наведені методи криптоаналізу систематизовані за хронологією їх появи і застосування для злому різних категорій криптосистем [52, 53]. Горизонтальна вісь поділена на часові проміжки: в область «вчора» потрапили атаки, які успішно використовувались для злому шифрів в минулому; «сьогодні» – методи криптоаналізу, що становлять загрозу для широко використовуваних в даний час криптосистем; «завтра» – ефективно застосовувані вже сьогодні методи, значення яких в майбутньому може зрости, а також методи, які поки не спричинили серйозного впливу на криптологію, однак з часом можуть створити прорив у зломах шифрів. На вертикальній осі позначені області застосування методів криптоаналізу: для злому криптосистем з таємним ключем, відкритим ключем або хеш-функцій.

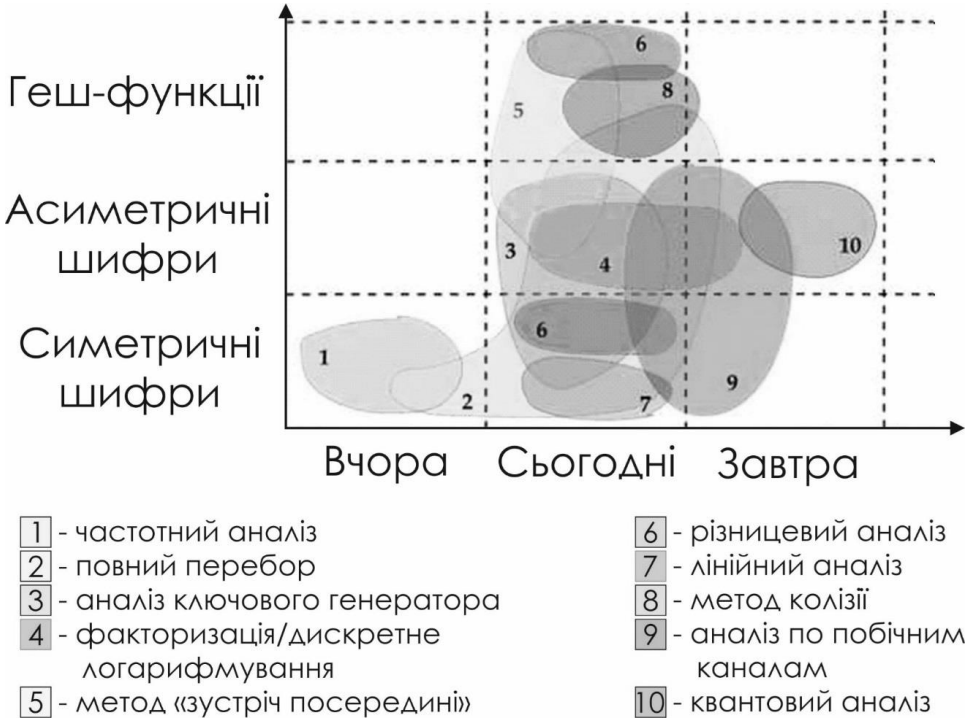


Рисунок 1.12 – Методи криптоаналізу

Загальні відомості. Найбільш важливою властивістю шифру є його *стійкість*, що характеризує його здатність протистояти атакам зловмисника, який зазвичай має намір визначити секретний ключ або відкрите повідомлення [37]. Загальноприйнятий принцип, що використовується при аналізі захищеності інформації, - це *правило Керкгоффса*, в якому сказано, що опис системи захисту інформації може бути відомий зловмиснику, а її стійкість ґрунтується тільки на невідомості секретного ключа [37]. Якщо метод шифрування невідомий, то до ключа можна додати ще декілька бітів, що будуть відповідати за його вибір [16].

В рамках правила Керкгоффса існує декілька типів (сценаріїв) атак [16]: атака за відомим шифротекстом (ciphertext-only attack), атака за відомим відкритим текстом (chosen-plaintext attack), атака за вибраним шифротекстом (chosen-ciphertext attack), атака за адоптивно обраним відкритим теккстом/шифротекстом (adaptive chosen-ciphertext/plaintext attack), атака на

пов’язані ключі (related-key attack), атака на пов’язані шифри (related-cipher attack) [54].

Необхідною вимогою до блочного шифру є його стійкість до універсальних атак, що забезпечується вибором його параметрів. Вважається, що стійкість проти метода повного перебору ключів гарантується при довжині ключа не менше 80 біт, а в найвідповідальніших застосуваннях – 256 біт. Для забезпечення стійкості шифру до атаки зі словником і до атаки, заснованої на парадоксі днів народження, слід обирати достатній розмір блоку, зазвичай рівний 128 біт [54, 55].

Різнисний метод і його різновиди. Різнисний метод [56] разом із своїми модифікаціями [23, 56] є поширеним методом оцінювання захищеності інформації при використанні блочних шифрів. Основною метою цього метода є пошук таких відмінностей блоків відкритого тексту, які могли б призвести до певної різниці блоків шифрованого тексту з відносно великою вірогідністю. Основним поняттям цього методу є різність, характеристика, диференціал, а також вірогідність характеристики і диференціалу.

Визначення 1.1. [57]. Нехай $X \in \{0,1\}^s$ і $Y \in \{0,1\}^s$, де $s > 0$, - деякі двійкові вектора. *Різністю* цих векторів називається результат операції *XOR*, викиненою над ними: $X \oplus Y$.

Визначення 1.2. [57]. *R-раундова різнісна характеристика* – це впорядкована множина $\Delta = (\delta^0, \ldots, \delta^R)$, де $\delta^r \in \{0,1\}^s$, $r = \overline{1, R}$. Для позначення характеристики використовується загальноприйняте позначення

$$\delta^0 \to \delta^1 \to \delta^2 \to \ldots \to \delta^{R-1} \to \delta^R.$$

Визначення 1.3 [57]. Нехай E – R-раундовий блочний шифр, Δ - R-раундова характеристика і $p \in [0,1]$. Будемо казати, що шифр E *допускає характеристику* Δ *з вірогідністю* p, якщо при випадково і незалежно обраних раундових ключах і блоках відкритого тексту C^0 і D^0 виконується

$$P\left(C^1 \oplus D^1 = \delta^1, \ldots, C^R \oplus D^R = \delta^R \mid C^0 \oplus D^0 = \delta^0\right) = p.$$

Визначення 1.4. [57]. R-раундовий диференціал – це впорядкована трійка $\left(\delta^{in}, \delta^{out}, R\right)$, де $\delta^{in} \in \{0,1\}^s$ і $\delta^{out} \in \{0,1\}^s$. Для позначення диференціала використовується наступний загальноприйнятий запис:

$$\delta^{in} \xrightarrow{R\text{раунд}} \delta^{out}.$$

Визначення 1.5. [57]. Нехай E – R-раундовий блочний шифр з розміром блоку s і $p \in [0,1]$. Будемо казати, що E *допускає диференціал* $\left(\delta^{in}, \delta^{out}, R\right)$ *з вірогідністю* p, якщо при випадково і незалежно обраних раундових ключах і блоках відкритого тексту C^0 і D^0 виконується рівність

$$P\left(C^R \oplus D^R = \delta^{out} \mid C^0 \oplus D^0 = \delta^{in}\right) = p.$$

Аналіз на основі заборонених диференціалів. В роботі [53] запропонована *атака на основі заборонених різниць* на шифр Skipjack, що дозволяє диференціали з нульовою вірогідністю.

Нехай деякий шифр E складається з R раундів і шифр $E^{[\overline{1,R-1}]}$ дозволяє диференціал $\delta_{inp} \to \delta_{out}$ з вірогідністю, що дорівнює нулю; в такому випадку необхідно зашифрувати певну кількість відкритих текстів з різницею δ_{inp}, а потім перебрати всі можливі ключі останнього раунда і розшифровувати для цього раунду отримані шифротексти, відкидаючи підключі, що приводять до різниці δ_{out}.

Заборонені диференціали часто будуються на основі підходу «розбіжність в середині» (miss-in-the-middle) [58]. Нехай $E = E_1 \circ E_2$, при цьому, шифр E_1 дозволяє диференціал $\delta_{inp} \to \delta_1$ з вірогідністю 1, а шифр E_2^{-1} дозволяє диференціал $\delta_{out} \to \delta_2$ також з вірогідністю 1. Якщо $\delta_1 \neq \delta_2$, диференціал $\delta_{out} \to \delta_{out}$ заборонений для шифру E.

Атака бумерангом. Нехай $E = E_1 \circ E_2$, при цьому, E_1 дозволяє диференціал $\alpha \xrightarrow[p]{} \beta$, а шифр E_2 дозволяє диференціал $\gamma \xrightarrow[q]{} \delta$, тоді алгоритм *атака бумерангом* (boomerang) [59] наступний:

- $C_1 := E(P_1)$ і $C_2 := E(P_2)$, де $P_1 \oplus P_2 = \alpha$;
- $P_3 := E^{-1}(C_1 \oplus \delta)$ і $P_4 := E^{-1}(C_2 \oplus \delta)$
- перевірити умову $P_3 \oplus P_4 = \alpha$.

Вірогідність події $P_3 \oplus P_4 = \alpha$ рівна $p^2 q^2$, і, якщо $pq > 2^{s/2}$, то шифр виявляється вразливим. Атака бумерангом реалізується за сценарієм адоптивно вибраного шифртексту. В роботах [57, 59] запропоновані її вдосконалення і метод конвертації даної атаки в атаку за вибраним відкритим текстом, що називається *посилена* (amplified) атака бумерангом.

Лінійний метод [60].

Визначення 1.6. [61] Нехай E – деякий блочний шифр з розміром блоку s і розміром розгорнутого ключа k (в бітах). Нехай $\vec{P} \in \{0,1\}^s$, $\vec{C} \in \{0,1\}^s$, $\vec{K} \in \{0,1\}^k$ - деякі двійкові вектори. Тоді *лінійною апроксимацією* для блочного шифру E називається умова

$$\vec{P} \cdot P \oplus \vec{C} \cdot C = \vec{K} \cdot K \tag{1.1}$$

яку позначимо як $\Omega(\vec{P}, \vec{C}, \vec{K})$, що пов'язує блок відкритого тексту P, блок шифрованого тексту C і розгорнутий ключ K. Тут $\vec{P} \cdot P = \bigoplus_{i=0}^{s} \vec{P^{[i]}} P^{[i]}$, $\vec{C} \cdot C = \bigoplus_{i=0}^{s} \vec{C^{[i]}} C^{[i]}$, $\vec{K} \cdot K = \bigoplus_{i=0}^{k} \vec{K^{[i]}} K^{[i]}$. Вектори $\vec{P}$, $\vec{C}$ і $\vec{K}$ називаються *лінійними масками*.

Позначимо як $p(\vec{P}, \vec{C}, \vec{K}, K)$ ймовірність того, що при випадково обраному блоку відкритого тексту $P \in \{0,1\}^s$ і відповідному йому блоку шифрованого

тексту $C = E(P)$ виконуються відношення (1.1), якщо шифрування виконується з розвернутим ключем K.

Визначення 1.7. [61] Ймовірність лінійної апроксимації $\Omega(\vec{P},\vec{C},\vec{K})$ - це середня по всім можливим ключам K ймовірність $p(\vec{P},\vec{C},\vec{K},K)$. Позначимо цю ймовірність як $p(\vec{P},\vec{C},\vec{K})$.

В [60] вказується на те, що в роботі [61], де запропонований лінійний аналіз, неявно робляться деякі припущення, зокрема, мається на увазі, що «майже для всіх ключів» $p(\vec{P},\vec{C},\vec{K},K) \approx p(\vec{P},\vec{C},\vec{K})$. Експериментальні і теоретичні результати [61] підтверджують доцільність цього припущення.

Тепер розглянемо, як використовується лінійна апроксимація для визначення ключа шифру. Отже, нехай $E = E^{[\overline{1,R-1}]} \circ E^{[R]}$ - R-раундовий блочний шифр, до того ж для $E^{[\overline{1,R-1}]}$ відома лінійна апроксимація $\Omega(\vec{P},\vec{C},\vec{K})$ з деякою ймовірністю p такою, що $|p - 1/2| \neq 0$. В такому випадку на цей шифр можливо розробити атаку за відомим відкритим текстом. Кількість необхідних для атаки блоків тексту (позначимо цю кількість як N) пропорційна $1/(p-1/2)^2$. Базовий алгоритм атаки, що вираховує ключ R-ого раунду K_R розміром k_R має наступний вигляд:

1. Візьмемо множину з N пар блоків відкритого і шифрованого тексту (P_i, C_i), $i = \overline{1,N}$. Кожному кандидату $K_R^{(j)}$, $j = \overline{0,2^{k_R}-1}$, для ключа K_R поставити у відповідність лічильник T_j, що дорівнює числу пар (P_i, C_i), $i = \overline{1,N}$, при яких ліва частина виразу (1.1) дорівнює 0.
2. Нехай $T_{\max}$ і $T_{\min}$ - це відповідно максимум і мінімум серед всіх T_j, а $j_{\max}$ і $j_{\min}$ - це відповідні їм номера ключів-кандидатів.
   - Якщо $|T_{mzx} - N/2| > |T_{\min} - N/2|$, тоді $K_R := K_R^{(i_{\max})}$ і права частина виразу (1.1) дорівнює 0 (якщо $p > 1/2$) або 1 (якщо $p < 1/2$).

- Якщо $|T_{mzx} - N/2| < |T_{\min} - N/2|$, тоді $K_R := K_R^{(i_{\min})}$ і права частина виразу (1.1) дорівнює 1 (якщо $p > 1/2$) або 0 (якщо $p < 1/2$).

При побудові лінійної апроксимації для шифру, як правило, будуються, а потім об'єднуються апроксимації для окремих раундів.

Визначення 1.8. [61] Нехай шифр E представимо в вигляді композиції $E = E_1 \circ \ldots \circ E_R$; K_r, $r = \overline{1,R}$, – ключі раундів, а розгорнутий ключ K є їх конкатенацією; $\Omega\left(\overrightarrow{P_r}, \overrightarrow{C_r}, \overrightarrow{K_r}\right)$, $r = \overline{1,R}$, є лінійними апроксимаціями для E_r, до того ж,

$$\overrightarrow{C_r} = \overrightarrow{P_{r+1}},\ r = \overline{1,R-1}. \tag{1.2}$$

Толі *об'єднанням* апроксимацій $\Omega\left(\overrightarrow{P_r}, \overrightarrow{C_r}, \overrightarrow{K_r}\right)$, $r = \overline{1,R}$, називається апроксимація

$$\Omega\left(\overrightarrow{P_1}, \overrightarrow{C_R}, \bigoplus_{i=1}^{R} \overrightarrow{K_i}\right). \tag{1.3}$$

Для обчислення ймовірності об'єднання апроксимацій використовується

Лема. [61] Нехай незалежні випадкові величини X_i, $i = \overline{1,N}$, мають розподіл Бернуллі з параметром $1 - p_i$, відповідно, $p_i \in [0,1]$, інакше кажучи, $P(X_i = 0) = p_i$ і $P(X_i = 1) = 1 - p_i$. Тоді

$$P(X_1 \oplus \ldots \oplus X_N = 0) = 1/2 + 2^{N-1} \prod_{r=1}^{N} (p_r - 1/2) \tag{1.4}$$

В припущенні, що робиться в [127], о незалежності і рівноймовірності вибору раундових ключів ймовірність об'єднання лінійних апроксимацій вираховується за формулою (1.4), де p_i - це ймовірність окремих апроксимацій. В цьому випадку належить задати

$$X_i = \vec{P} \cdot P \oplus \vec{C} \cdot C \oplus \vec{K} \cdot K$$

і скористатись лемою. Оскільки виконується умова (1.2), то умова $X_1 \oplus \ldots \oplus X_R = 0$ можливо перетворити до вигляду (1.3).

Інтегральний метод.

Інтегральний аналіз [62] – це узагальнення ряду методів, що використовуються для розробки алгоритмів вирахування секретного ключа шифрів Square, Misty1 і Misty2, Twofish і узагальнення структури SASAS [23]. Автори використовували різні назви, що підкреслюють особливості атак: *квадратна атака* (square), *насичена атака* (saturation), *мультімножина атака* (multiset), *структурний аналіз* (structural). Застосування інтегрального аналізу передбачає дослідження так званих мультімножин (табл. 1.7) з метою побудови інтегральної характеристики, яка показує, як змінюються їх властивості впродовж раундів.

Таблиця 1.7

Типи мультимножин

C	Константна (Constant)	Всі елементи множини однакові
A	Насичена (All)	Всі елементи множини різні (всі 2^m елементів довжини m біт)
E	Парна (Even)	Кожний елемент зустрічається парну кількість раз
B	Збалансована (Balanced)	Сума (XOR) всіх елементів дорівнює нулю
D	Подвійна (Dual)	Або насичена, або парна

Як приклад розглянемо інтегральну характеристику шифру Rijndael (див. рис. 1.10), що описується в [62]. Уявімо 128-бітний (16-байтний) блок цього шифру в вигляді квадрата 4×4, де кожна клітина відповідає одному байту. Візьмемо множину, що складається з 256 блоків, таких що перший байт цих блоків приймає всі можливі 256 значень, а інші байти – однакові; таким чином, множина перших байт є множиною типу ***A***, а інші – ***C***. Побудована інтегральна характеристика, дозволяє вирахувати ключ шифру, що складається з чотирьох раундів.

Статистичні методи. В режимі CTR і OFB блочний шифр може використовуватись в якості генератора псевдовипадкових чисел. Розглянемо застосування випадкових чисел, види генераторів псевдовипадкових чисел, а також зазначимо актуальність задачі статистичного аналізу блочних шифрів.

Одним з стійких генераторів псевдовипадкових чисел – це режим блочного шифру CTR (рис. 1.13), заданий наступними формулами [63]:

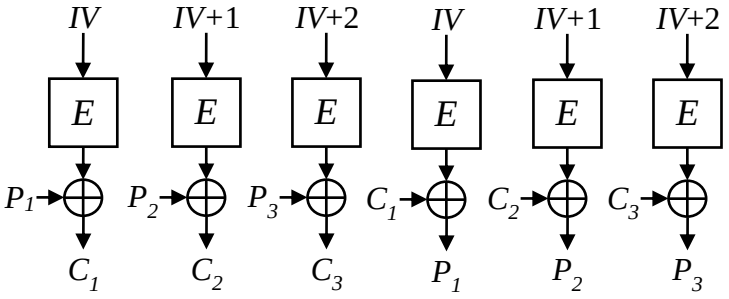


Рисунок 1.13 – Режим лічильника (CTR)

Зашифрування:	Розшифрування:
$Z_0 = IV$;	$Z_0 = IV$;
$C_i = P_i \oplus E_K(Z_i)$;	$P_i = C_i \oplus E_K(Z_i)$;
$Z_{i+1} = Z_i + 1$.	$Z_{i+1} = Z_i + 1$.

Тут IV - це початкове значення, P_i і C_i - це відповідно блоки відкритого і шифрованого тексту, а E_K - це функція зашифрування з ключем K. Для генерації псевдовипадкових чисел слід використовувати формулу для зашифрування, тоді біти блоків C_i модулюють випадкові числа, що мають розподіл Бернулі з параметром 1/2.

Поточні шифри вже самі собою є генераторами псевдовипадкового ключового потоку, тому отримані байти модулюють рівномірно розподілені випадкові біти.

Перевірка якості псевдовипадкових чисел виконується за допомогою статистичних тестів і критеріїв, деякі з яких вже стали класичними [63]: критерій хі-квадрат, критерій Колмогорова, критерій Смірнова, критерій

Пірсона, спектральний тест, інші є відносно новими: адаптивний критерій хі-квадрат, порядковий тест, тест «стопка книг» і деякі тести, рекомендовані НІСТ США. Ряд тестів основан на ідеях теорії інформації, наприклад, тест Маурера, методи на основі кодів Лемпела-Зіва [64].

Інші методи криптоаналізу.

Алгебраїчний аналіз [63] передбачає представлення шифру в вигляді системи рівнянь, відкритий текст, шифрований текст і ключ. Потім пропоується підставити деяке число пар блоків відкритого і шифрованого тексту в систему і вирішити її.

Інтерполяційний аналіз [63] має на увазі інтерполяції (наприклад, за допомогою многочлена) для компонентів шифру з подальшим визначенням невідомих коефіцієнтів методами чисельного аналізу. При побудові наближуючої функції точками, по яким вона будується, використовуються пари блоків відкритого і шифрованого тексту.

Атака за допомогою бічних каналів (side-channel) спрямовані не на теоретичне дослідження шифру, а на дослідження його фізичних реалізацій. Для проведення даного типу атак зловмисник повинен мати доступ до пристрою шифрування з метою замірювання яких-небудь характеристик: часу шифрування або реалізації певних операцій, споживаємої потужності, електромагнітного випромінювання. Якщо яка-небудь з наведених характеристик процеса шифруваня залежить від секретного ключа, то його можна вирахувати. Атаки за допомогою бічних каналів поділяються на тимчасовий (timing) аналіз, простий аналіз потужності (simple power), різнисний аналіз потужності (differential power), простий електромагнітний аналіз (simple electromagnetic) і різнисний електромагнітний аналіз (differential electromagnetic) [23, 63].

Ковзна атака [63] може бути реалізована в рамках моделі за відомим відкритим текстом зі складністю $O\left(2^{s/2}\right)$, де s - це розмір блоку. Для шифрів

Фейстеля (в них за один раунд перетворюється тільки половина блоку) ця атака може буть реалізована в рамках моделі за вибраним відкритим тестом зі складністю $O\left(2^{s/4}\right)$. Для того щоб до шифру можна було б застосувати цю атаку, він має складатися з однакових раундів, і раундова функція F повинна бути *слабкою* (ключ одноразового шифру може буть «легко» знайдений).

1.4. Паралельні обчислення в тривимірному просторі

Розподілені обчислення - спосіб вирішення трудомістких обчислювальних завдань з використанням декількох комп'ютерів, найчастіше об'єднаних в паралельну обчислювальну систему [65]. Послідовні обчислення в розподілених системах виконуються з урахуванням одночасного вирішення багатьох завдань.

Паралельні обчислення - спосіб організації комп'ютерних обчислень, при якому програми розробляються як набір взаємодіючих обчислювальних процесів, що працюють паралельно (одночасно). Термін охоплює сукупність питань паралелізму в програмуванні, а також створення ефективно діючих апаратних реалізацій. Теорія паралельних обчислень становить розділ прикладної теорії алгоритмів [65].

Паралельні програми можуть фізично виконуватися або послідовно на єдиному процесорі - перемежовуючи по черзі кроки виконання кожного обчислювального процесу, або паралельно - виділяючи кожному обчислювальному процесу один або кілька процесорів (що знаходяться поруч або розподілених в комп'ютерну мережу).

Основна складність при проектуванні паралельних програм - забезпечити правильну послідовність взаємодій між різними обчислювальними процесами, а також координацію ресурсів, поділюваних між процесами.

Надзвичайна паралельність (надзвичайно паралельна задача, англ. Embarrassingly parallel) - тип завдань в системах паралельних обчислень, для

яких не потрібно докладати великих зусиль при поділі на кілька окремих паралельних завдань (розпаралелюванні). Найчастіше не існує залежності (або зв'язку) між цими паралельними завданнями, тобто їх результати не впливають один на одного. [65]

Надзвичайно паралельні завдання практично не вимагають узгодження між результатами виконання окремих етапів, що відрізняє їх від завдань розподілених обчислень, які вимагають зв'язку проміжних результатів. Паралельні завдання легкі для виконання на серверних фермах (серверних кластерах), вони добре підходять для великих розподілених платформ в Інтернеті, таких як BOINC.

Типовим прикладом надзвичайно паралельної задачі є робота графічного процесора (GPU) при розрахунку 3D проекцій, коли кожен піксель на екрані може розраховуватися самостійно.

1.5. Висновки до першого розділу

Розглянуто базові поняття шифрування, до яких входять: Режими шифрування; Поширення помилок; Вибір режиму шифрування; Задачі криптографії; Криптографічні примітиви; Алгоритми шифрування; Ітеративні блочні шифри і їх структура і принципи побудови; Стандартизація і застосування блочних шифрів.

Проаналізована класифікація та надана коротка характеристика алгоритмів криптографічного захисту інформації, що включає в себе опис Формування міжнародних стандартів шифрування; ГОСТ 28147; Rijndael (AES); ДСТУ 7624:2014 («Калина»).

Показані основні атаки на шифри та критерії ефективності алгоритмів СБШ. Наведено основні поняття криптоаналізу. Розглянуто такі методи криптоаналізу: Різнисний метод і його різновиди; Лінійний метод; Інтегральний метод; Статистичні методи та ін.

Крім наведеною раніше класифікацією алгоритмів шифрування, згідно з якою розрізняють бесключевые (хеш-функції) шифри, одноключевые (симетричні) і двохключові (асиметричні) шифри, алгоритми шифрування можуть бути класифіковані також за ознакою розмірності простору, в якому представляються дані, що підлягають криптографічним перетворенням, і ключі. За цією ознакою будемо розрізняти 1D- (Dimension), 2D - і 3D-шифри. До 1D - віднесемо шифри, в яких дані і ключі представлені у вигляді одновимірних векторів. В якості представника такого підкласу шифрів можемо вказати, наприклад, DES-шифр. У той же час сучасний AES стандарт симетричного блокового шифрування, заснований на алгоритмі Rijndael, є класичним представником підмножини 2D-шифрів. Теорія синтезу і аналізу (оцінки криптостійкості) 1D - 2D-шифрів до теперішнього часу досить добре розвинені, тоді як теорія 3D-шифрування знаходиться у зародковому стані і не доведена ще до практичної реалізації. У силу зазначених причин напрямок 3D-шифрування, яке покладено в основу дисертаційного дослідження, є, безумовно, актуальним і відкриває нову сторінку в теорії і практиці криптології.

Таким чином, у першому розділі проведено аналіз сучасних криптографічних методів захисту інформації за критеріями стійкості, швидкодії та необхідних ресурсів для проведення перетворень, а також обґрунтовані шляхи вдосконалення алгоритмів симетричного шифрування, розглянуто вітчизняну та зарубіжну літературу за темою дисертаційної роботи. Визначено основні поняття, пов’язані із криптографічним захистом інформації, проаналізовано основні завдання, які може розв’язувати криптографія у процесі захисту інформації. Крім того, формалізовано вимоги до сучасних БШ (на основі критеріїв міжнародного (AES), європейського (NESSIE) та вітчизняного конкурсів).

Виявлені переваги і недоліки розповсюджених БШ. ГОСТ 28147-89 хоч і забезпечує практичну стійкість, але для нього вже розроблені теоретичні

методи криптоаналізу. Зараз його замінюють шифри СТБ 34.101.31-2007, ДСТУ 7624:2014, ГОСТ Р 34.12-2015. Оскільки AES реалізовано в багатьох мікросхемах на апаратному рівні, то дедалі більше підлягають сумніву пристрої, захищені AES на апаратному рівні, оскільки проведені дослідження – це статті, у яких описано як саме реалізована уразливість у реалізаціях AES. Тому світові лідери IT-індустрії вже використовують нові алгоритми, наприклад Google в 2014 р. впровадив шифр ChaCha20 для захисту каналів зв'язку пристроїв на базі Android. ДСТУ 7624:2014 (БШ «Калина») розроблений під малоресурсну (lightweight) криптографію, яка передбачає компактну реалізацію і мінімальне енергоживлення, забезпечуючи прийнятний рівень криптостійкості і має порівняно малу швидкодію. Алгоритми AES і ДСТУ 7624:2014 основані на архітектурі «Square» (квадрат), що являє собою прямі перетворення шифрованих блоків, які представлені у вигляді двувимірного байтового масиву. Алгоритм ДСТУ 7624:2014 має більш повільну апаратну реалізацю. Алгоритм ГОСТ 28147-89 базується на архітектурі «мережа Файстеля», недоліком якої порівняно з використовуваною в алгоритмах AES і ДСТУ 7624:2014 є те, що за один раунд шифрується тільки половина блоку. Шифри AES і ДСТУ 7624:2014 використовують 2D-перетворення, а шифри 3DES і ГОСТ 28147-89 – 1D-перетворення.

Розглянуті ітеративні БШ і їх структура, принципи побудови, стандартизація і застосування БШ. Наведені способи оцінювання захищеності інформації під час використання БШ: диференціальний, лінійний, інтегральний, статистичний та інші методи криптоаналізу.

РОЗДІЛ 2
МАТЕМАТИЧНІ ОСНОВИ ТРИВИМІРНИХ ОБЧИСЛЕНЬ

В класичній криптографії використовується одномірні і двомірні криптографічні перетворення. В останні роки з'явилась персональні комп'ютери (ПК) з багатоядерними центральними і графічними процесорами (ЦП і ГП). Також з'явились середовища програмування що дають можливість паралельного програмування. Це дає можливість створювати багатопоточні, побудовані на багатовимірних перетвореннях, методи криптографічного перетворення. Раніше такої можливості для пересічного користувача не було. Під тривимірні перетворення в сучасних ГП і ПК розроблені апаратні рішення, що робить багатовимірні розрахунки такими ж швидкими як одно- і двомірна обробка даних. Зараз ведуться розробки носіїв інформації, що орієнтовані на зберігання тривимірних даних. Просту тривимірну структуру можна представити як складну двомірну, або як дуже складно одномірну (а обробка складних структур потребує багато часу). То постає задача для аналізу існуючих і синтезу нових методів обробки тривимірних структур. Найочевидніша тривимірна структура, це тривимірна матриця, яка синергетично поєднує в собі можливості векторів і звичайних двовимірних матриць, добавляючи при цьому ще й нові властивості, які можна використати в криптографії.

2.1. Основні поняття та визначення.

Тривимірні перетворення. В однорідних координатах точка в тривимірному просторі з координатами (x, y, z) записується як $(W \cdot x, W \cdot y, W \cdot z, W)$, де $W <> 0$. Будемо вважати, що $W = 1$. Тривимірні перетворення в однорідних координатах представляються матрицями розміром 4×4. Тривимірне переміщення є простим розширенням двомірного і описується матрицею [69]:

$$T(D_x, D_y, D_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D_x & D_y & D_z & 1 \end{bmatrix} \quad (2.1)$$

При цьому:

$$[x, y, z,1] \cdot T(D_x, D_y, D_z) = \lfloor x + D_x, y + D_y, z + D_z,1 \rfloor \quad (2.2)$$

Перетворення масштабування [66] також є простим розширенням двомірного і описується матрицею:

$$S(S_x, S_y, S_z) = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.3)$$

При цьому:

$$[x, y, z,1] \cdot T(S_x, S_y, S_z) = \lfloor x \cdot S_x, y \cdot S_y, z \cdot S_z,1 \rfloor \quad (2.4)$$

Розглянемо операцію повороту. Так як при двомірному повороті [66] в площині координати залишаються незмінні, то поворот навколо осі z записується так:

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 & 0 \\ -\sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.5)$$

Матриця повороту навколо осі x [66]:

$$R_z(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.6)$$

і навколо осі y [66]:

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.7)$$

Зворотні матриці будуть виражатись зворотними матрицями. Для операції переносу потрібно лише замінити знаки компонент вектору переносу на протилежні [66]:

$$T^{-1}(D_x, D_y, D_z) = T(-D_x, D_y, D_z) \quad (2.8)$$

для операції масштабування – на зворотні значення:

$$S^{-1}(S_x, S_y, S_z) = S(1/S_x, 1/S_y, 1/S_z) \quad (2.9)$$

для повороту – обиранням від'ємного кута повороту:

$$R(\alpha) = R(-\alpha) \quad (2.10)$$

Результатом декількох послідовних поворотів буде матриця [66]:

$$A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.11)$$

Стовбці і строки підматриці 3×3 матриць повороту R_x, R_y, R_z, аналогічно двомірному випадку, являють собою взаємно ортогональні одиничні вектори. Ця матриця називається ортогональною. Важливою її властивістю є те, що зворотна до неї матриця є транспонованою:

$$B^{-1} = B^T \quad (2.12)$$

Після перемноження будь якої кількості матриць виду, результуюча матриця завжди буде мати вигляд:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{2.13}$$

Тут верхня частина розміром визначає сумарний поворот і масштабування, а три коефіцієнти останнього рядка – сумарне переміщення. З точи зору швидкодії розрахунків виконувати перетворення вигідно по формулі:

$$[x', y', z'] = [x, y, z] \cdot R + T \tag{2.14}$$

Перетворення симетрії відносно заданої площини. Деякі орієнтації тривимірних об’єктів неможливо отримати одними обертаннями, потрібні перетворення відображення. В тривимірному просторі відображення відбувається відносно площини. За аналогією з двовимірним відображенням, тривимірне відображення відносно площини еквівалентно обертанню навколо осі в тривимірному просторі в чотиривимірне і назад в початковий тривимірний простір. Для чистого відображення детермінант матриці дорівнює -1.

В загальному випадку матриця відображення має наступний вигляд [69]:

$$[R'] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ – відображення відносно } xy. \tag{2.15}$$

$$[R'] = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ – відображення відносно } yz. \tag{2.16}$$

$$[R'] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ – відображення відносно } xz. \qquad (2.17)$$

Симетрії відносно осей x, y, z і точки 0 (початок координат) задаються матрицями

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.18)$$

Зсув реалізується складанням, а масштабування і поворот – множенням.

Перетворення масштабуванням [66] (дилатація) відносно початку координат має вигляд:

$$X_n = X \cdot D_x, \quad Y_n = Y \cdot D_y, \quad Z_n = Z \cdot D_z \qquad (2.19)$$

або в матричній формі: $P_n = P \cdot D$, де D_x, D_y, D_z – коефіцієнти масштабування по осям, а $D = \begin{bmatrix} D_x & 0 & 0 & 0 \\ 0 & D_y & 0 & 0 \\ 0 & 0 & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ – матриця масштабування.

При $D > 1$ відбувається розширення, при $0 \le D < 1$ стискання.

Перетворення повороту відносно початку координат має вигляд [66]:

$$\begin{aligned} X_n &= X \cdot \cos\alpha - Y \cdot \sin\alpha, \\ Y_n &= X \cdot \cos\alpha + Y \cdot \cos\alpha. \end{aligned} \qquad (2.20)$$

або в матричній формі: $P_n = P \cdot R$

де α – кут повороту, а

$$R=\begin{bmatrix}\cos\alpha & \sin\alpha & 0\\ -\sin\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix} \text{ – матриця повороту.} \qquad (2.21)$$

Зауваження: Стовбці і строки матриці повороту представляють собою взаємно ортогональні одиничні вектори. Дійсно квадрати довжин векторів-строк рівні одиниці:

$$\cos\alpha\cdot\cos\alpha+\sin\alpha\cdot\sin\alpha=1 \text{ і } (-\sin\alpha)\cdot(-\sin\alpha)+\cos\alpha\cdot\cos\alpha=1 \qquad (2.22)$$

а скалярний добуток векторі-строк це

$$\cos\alpha\cdot(-\sin\alpha)+\sin\alpha\cdot\cos\alpha=0 \qquad (2.23)$$

Оскільки скалярний добуток векторів $\mathrm{A}\cdot\mathrm{B}=|\mathrm{A}|\cdot|\mathrm{B}|\cdot\cos\psi$, де $|\mathrm{A}|$ – довжина вектору A, $|\mathrm{B}|$ – довжина вектору B, а ψ – найменший додатній кут між ними, то з рівності 0 скалярного добутку двох векторів-строк довжини 1 випливає, що кут між ними дорівнює $90°$.

Перетворення зсуву [66]. Матриця зсуву має вигляд:

$$T=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ T_x & T_y & T_z & 1\end{bmatrix} \qquad (2.24)$$

де T_x, T_y, T_z – величина зсуву по осям.

Матриця зворотного перетворення для зсуву утворюється шляхом зміни знаку у T_x, T_y, T_z.

Зауваження: слід пам’ятати, що результат перетворення залежить від послідовності виконання операцій.

Структура тривимірних матриць. Будь яка система з n^3 елементів $A_{ijk}\,(i,j,k=1,2,...,n)$ поля P, розташованих в точках тривимірного простору, що визначається координатами i,j,k називається *тривимірною (кубічною)*

матрицею n-го порядку над P. Наприклад, система 2^3 елементів $A_{ijk}\,(i, j, k = 1,2)$, розташованих в вигляді куба (рис.2.1), являє собою кубічну матрицю 2-го порядку.

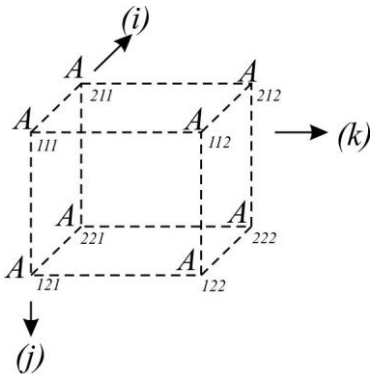


Рисунок 2.1 – Тривимірна (кубічна) матрицю 2-го порядку

Кубічну матрицю *n*-го порядку зі спільним елементом A_{ijk} будем позначать скорочено символом [70]

$$\left\|A_{ijk}\right\| \ (i, j, k = 1,2,...,n), \tag{2.25}$$

а в тих випадках, коли це не викликає непорозумінь, просто як A. Якщо індекси i, j, k приймають відповідні значення $i = 1,2,...,l$, $j = 1,2,...,m$, $k = 1,2,...,n$ при l, m, n різних між собою, то тривимірна матриця, що складається з lmn елементів, представляється в вигляді прямокутного паралелепіпеда.

Сукупність елементів матриці (2.25) з фіксованим значенням індексу i називається *перетином орієнтації* (i). [68] Всі n перетинів орієнтації (i) в матриці (2.25) паралельні один одному і є звичайними, двомірними матрицями *n*-го порядку

$$\left\|A_{1jk}\right\|, \left\|A_{2jk}\right\|, \ldots, \left\|A_{njk}\right\| \ \ (i, j, k = 1,2,...,n).$$

Аналогічне визначаються перетини орієнтацій (j) і (k).

Сукупність елементів матриці (2.25) з фіксованим значенням індексів j, k називається *перетином (двократним) орієнтації* (jk) *або строчкою напрямку*

(i). [68] Всі n^2 строк цього напрямку паралельні одна одній і це фактично одномірні матриці n-го порядку

$$\left\| A_{1\bar{j}\bar{k}}, A_{2\bar{j}\bar{k}}, ..., A_{n\bar{j}\bar{k}} \right\|,$$

де $\bar{j}, \bar{k}$ – фіксовані значення індексів j, k. Аналогічно визначаються строки напрямків (j) i (k).

В кубічній матриці n-го порядку два перетини різної орієнтації мають n спільних елементів, розташованих в одній строчці, тоді як три перетини різних орієнтацій мають тільки один спільний елемент. Кожен перетин довільної орієнтації і кожна строчка напрямку, що перпендикулярна до цього перетину, мають один і тільки один спільний елемент. *Відповідними* елементами двох паралельних перетинів називаються елементи, що належать одній і тій самій строчці, перпендикулярній до цього перетину. *Відповідними* елементами двох паралельних строк називаються елементи, що належать одному і тому ж перетину, що перпендикулярний до цих строк [69].

Використовуючи двовимірні перетини, можна записати просторову матрицю в вигляді квадратної або прямокутної таблиці в залежності від того, буде число вимірів матриці парним чи непарним; двовимірні перетини при цьому відділяються один від одного вертикальною або горизонтальною лінією. Наприклад, кубічна матриця 2-го порядку (рис.2.25) за допомогою перетинів орієнтації (i) може бути записана в вигляді прямокутника

$$\left\| \begin{array}{cc|cc} A_{111} & A_{112} & A_{211} & A_{212} \\ A_{121} & A_{122} & A_{221} & A_{222} \end{array} \right\| \begin{array}{l} \longrightarrow (i) \\ \quad \downarrow \\ (j) \end{array} \begin{array}{l} \\ \longrightarrow (k) \\ \\ \end{array},$$

стрілки вказують напрямок, в якому збільшуються відповідні індекси.

Трансверсалі і діагоналі тривимірних матриць. Елементи просторової матриці A, взяті в кількості, що не перевищує її порядок n, називають, *трансверсальними*, якщо не одна пара їх не належить одному і тому ж перетину (простому) будь-якої орієнтації [68]. Сукупність n трансверсальних елементів

матриці A, представлених в вигляді одновимірної матриці *n*-го порядку, утворює *трансверсаль*. Наприклад, в кубічній матриці 3-го порядку (рис.2.2) сукупність елементів A_{112}, A_{231}, A_{323} утворюють одну з її трансверсалей, кількість яких дорівнює $(3!)^2$. В кубічній матриці *n*-го порядку нараховується $(n!)^2$ трансверсалей. Серед них знаходиться 4 *діагоналі*, утворених елементами, які розташовані на прямій, що з'єднують протилежні вершини матриці. Та з діагоналей, у якої в кожному елементі значення всіх індексів однакові, називається *головною*, а її перший елемент A_{111} – *головним*. Інші діагоналі називаються *другорядні* [69]. В матриці (рис.2.2) присутні 4 діагоналі

$$A_{111}, A_{222}, A_{333};\quad A_{113}, A_{222}, A_{331};\quad A_{131}, A_{222}, A_{313};\quad A_{133}, A_{222}, A_{311}$$

з них перша – голова діагональ з головним елементом A_{111}.

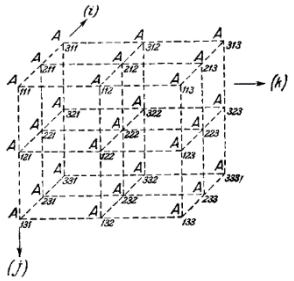


Рисунок 2.2 – Тривимірна (кубічна) матрицю 3-го порядку

2.2. Базові оператори тривимірної модулярної алгебри.

Транспоновані тривимірні матриці. Матрицю, що утворюється з тривимірної матриці

$$A = \left\| A_{i_1 i_2 i_3} \right\| \quad (i_1, i_2, i_3 = 1,2,...,n)$$

шляхом обміну значень двох яких небудь індексів, наприклад i_1 і i_2, у всіх її елементах, називатється *транспонованою відносно A по цим індексам* [69]. Позначимо її символом $A^{(i_1,i_2)}$. Елементи такої матриці пов'язані з елементами початкової матриці A співвідношеннями

$$A^{(i_1,i_2)}_{i_1 i_2 i_3} = A_{i_2 i_1 i_3} \qquad (i_1, i_2, i_3 = 1,2,...,n).$$

Так, наприклад, матриці $A^{(i,j)}$, $A^{(j,k)}$, $A^{(i,k)}$, транспоновані по двом індексам відносно кубічної матриці 2-го порядку (рис.2.1), мають відповідний вигляд (рис.2.3).

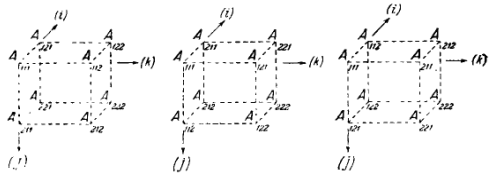


Рисунок 2.3 – Транспоновані матриці $A^{(i,j)}$, $A^{(j,k)}$, $A^{(i,k)}$ відносно A

Матрицю

$$A' = \left\| A'_{i_1 i_2 i_3} \right\| \ (i_1, i_2, i_3 = 1,2,...,n),$$

елементи якої пов'язані з елементами зазначеної вище матриці A співвідношенням

$$A'_{i_1 i_2 i_3} = A_{i_{\alpha_1} i_{\alpha_2} i_{\alpha_3}},$$

де i_{α_1}, i_{α_2}, i_{α_3} - яка небудь перестановка з індексів i_1, i_2, i_3, будем називати транспонованою відносно A відповідно до підстановки $\begin{pmatrix} i_1 & i_2 & i_3 \\ i_{\alpha_1} & i_{\alpha_2} & i_{\alpha_3} \end{pmatrix}$.

Цю матрицю будем позначати також символом

$$A^{\begin{pmatrix} i_1 & i_2 & i_3 \\ i_{\alpha_1} & i_{\alpha_2} & i_{\alpha_3} \end{pmatrix}}.$$

Кількість всіх матрицю, транспонованих відносно даної тривимірної матриці A, що містить і матрицю A, яку можна розглядати як транспоновану відповідно тотожній підстановці $\begin{pmatrix} i_1 & i_2 & i_3 \\ i_1 & i_2 & i_3 \end{pmatrix}$, очевидно дорівнює 3! або 6.

Альтернативні і неальтернативні індекси. Індекс i, значення якого обумовлюється додатнім або від'ємним знаком додаткового множника $(-1)^{I_i}$, називається *альтернативним*, а індекси j, k – *неальтернативними* (ці визначення відносяться не тільки до індексів, а також до строк і їх напрямків, до перетинів (простих) і їх орієнтацій та ін.) [69].

Кубічний детермінант і сигнатура. Для позначення *кубічного детермінанта n-го порядку з сигнатурой* $(\overset{+}{i}\overset{\pm}{j}\overset{\pm}{k})$, використовується символ Райса $\left|A_{\overset{+}{i}\overset{\pm}{j}\overset{\pm}{k}}\right|$, де між вертикальними лініями написан загальний елемент матриці (2.25) і над альтернативними індексами j, k поставлен знак $\pm$, а над неальтернативним індексом i знак $+$. Цю сигнатуру скорочено будем позначати в вигляді $(\overset{+}{i})$, вказуючи явно тільки неальтернативний індекс i [68].

Аналогічно визначаються *кубічні детермінанти n-го порядку* $\left|A_{\overset{\pm}{i}\overset{+}{j}\overset{\pm}{k}}\right|$ *і* $\left|A_{\overset{\pm}{i}\overset{\pm}{j}\overset{+}{k}}\right|$ *з сигнатурою* $(\overset{+}{j})$ *і* $(\overset{+}{k})$.

Сума $(n!)^2$ членів добутку елементів трансверсалі, утворених для всіх трансверсалей матриці (2.25), дає *кубічний детермінант n-го порядку* $\left|A_{\overset{+}{i}\overset{+}{j}\overset{+}{k}}\right|$ *з сигнатурою* $(\overset{+}{i}\overset{+}{j}\overset{+}{k})$, яка зазвичай називається *кубічним перманентом* матриці (2.1).

В кожному члені кубічного детермінанта з той або іншою сигнатурою його елементи, очевидно, можуть розташовуватись в такій послідовності, щоб перестановка $i^{(1)},i^{(2)},...,i^{(n)}$, що утворюється зі значень першого індексу i, представлена послідовністю натуральних чисел $1,2,...,n$.

Число що вказує порядок детермінанта, якщо потрібно звертати на це увагу, будем писати внизу правої вертикальної лінії в позначенні детермінанта.

Таким чином, кубічний детермінант 2-го порядку матриці (2.25)

$$\left|A_{\overset{+}{i}\overset{\pm}{j}\overset{\pm}{k}}\right|_2 = A_{111}A_{222} - A_{112}A_{221} - A_{121}A_{212} + A_{122}A_{211}, \qquad (2.26)$$

$$\left|A_{\overset{+}{i}\overset{+}{j}\overset{\pm}{k}}\right|_2 = A_{111}A_{222} - A_{112}A_{221} + A_{121}A_{212} - A_{122}A_{211}, \qquad (2.27)$$

$$\left|A_{\overset{\pm}{i}\overset{\pm}{j}k}\right|_2 = A_{111}A_{222} + A_{112}A_{221} - A_{121}A_{212} - A_{122}A_{211}, \qquad (2.28)$$

$$\left|A_{\overset{+}{i}\overset{+}{j}k}\right|_2 = A_{111}A_{222} + A_{112}A_{221} + A_{121}A_{212} + A_{122}A_{211}, \qquad (2.29)$$

можуть бути представлені в вигляді рис.2.4 [69]

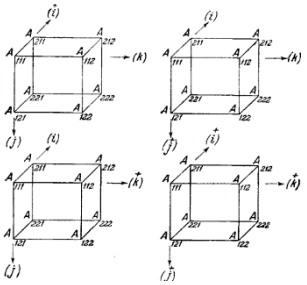


Рисунок 2.4 – кубічний детермінант 2-го порядку матриці A

або в вигляді

$$\begin{vmatrix} A_{111} & A_{112} \\ A_{121} & A_{122} \end{vmatrix} \begin{matrix} \overset{+}{\rightarrow}(i) \\ \downarrow \\ (j) \end{matrix} \begin{vmatrix} A_{211} & A_{212} \\ A_{221} & A_{222} \end{vmatrix} \rightarrow (k), \qquad \begin{vmatrix} A_{111} & A_{211} \\ A_{112} & A_{212} \end{vmatrix} \begin{matrix} \overset{+}{\rightarrow}(j) \\ \downarrow \\ (k) \end{matrix} \begin{vmatrix} A_{121} & A_{221} \\ A_{122} & A_{222} \end{vmatrix} \rightarrow (i),$$

$$\begin{vmatrix} A_{111} & A_{121} \\ A_{211} & A_{221} \end{vmatrix} \begin{matrix} \overset{+}{\rightarrow}(k) \\ \downarrow \\ (i) \end{matrix} \begin{vmatrix} A_{112} & A_{122} \\ A_{212} & A_{222} \end{vmatrix} \rightarrow (j), \qquad \begin{vmatrix} A_{111} & A_{112} \\ A_{121} & A_{122} \end{vmatrix} \begin{matrix} \overset{+}{\rightarrow}(i) \\ \downarrow \\ (j) \end{matrix} \begin{vmatrix} A_{211} & A_{212} \\ A_{221} & A_{222} \end{vmatrix} \rightarrow (k).$$

Детермінант і перманент звичайної матриці n-го порядку [69]

$$\left\|A_{ij}\right\| \quad (i, j = 1,2,...,n)$$

також можуть позначатись символами Райса

$$\left|A_{\overset{\pm}{i}\overset{\pm}{j}}\right|_n \text{ і } \left|A_{\overset{+}{i}\overset{+}{j}}\right|_n,$$

як двомірні детермінанти з сигнатурою $(\overset{\pm}{i}\overset{\pm}{j})$ і $(\overset{+}{i}\overset{+}{j})$.

Таким чином, маємо:

$$\left|A_{\overset{\pm}{i}\overset{\pm}{j}}\right|_2 = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} = A_{11}A_{22} - A_{12}A_{21},$$

$$\left|A_{\overset{+}{i}\overset{+}{j}}\right|_2 = \begin{vmatrix} \overset{(+)}{A_{11}} & \overset{(+)}{A_{12}} \\ A_{21} & A_{22} \end{vmatrix} = A_{11}A_{22} + A_{12}A_{21}$$

Додаткові множники вигляду $(-1)^N$, на які доводиться помножувати добуток елементів кожної трансверсалі кубічної матриці n-го порядку, щоб

отримати члени її детермінанта з той або іншою сигнатурою, легко знаходити *графічним способом* (рис.2.5) [68].

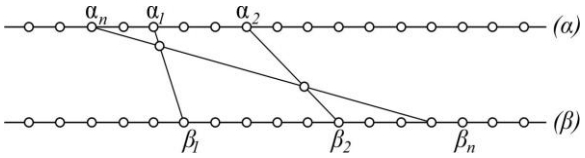


Рисунок 2.5 – Діаграма перестановок для двомірної матриці

Для пояснення цього способу використаємо цю діаграму, яка дозволяє визначити парність будь-якої підстановки *n*-го ступеня

$$\begin{pmatrix} \alpha_1\alpha_2...\alpha_n \\ \beta_1\beta_2...\alpha_n \end{pmatrix},$$

вона співпадає з парністю суми $I_\alpha + I_\beta$ чисел інверсії I_α і I_β в перестановках

$$\alpha_1,\alpha_2,...,\alpha_n,$$

$$\beta_1,\beta_2,...,\beta_n$$

з чисел $1,2,...,n$.

В діаграмі (рис.2.5) кожна пара елементів α_m, β_m $(m=1,2,...,n)$ підстановки порівнюється з відрізком, що з'єднує пару відповідних точок на горизонталях (α) і (β). Кількість перетинів побудованих таким чином n відрізків має, очевидно, ту ж парність, що і підстановка яку розглядаєм (точка перетину n відрізків вважається за C_n^2 перетинів).

Візьмем тепер три горизонталі (рис.2.6) і на кожній з них відмітимо n точок, що відповідають значенням $1,2,...,n$ індексів i, j, k, так, щоб точки з однаковими значеннями цих індексів лежали на одній і тій самій вертикалі (рис.2.6).

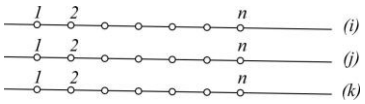


Рисунок 2.6 – Діаграма перестановок для тривимірної матриці

Тоді кожному елементу $A_{i^{(\alpha)}j^{(\alpha)}k^{(\alpha)}}$ $(\alpha=1,2,...,n)$ якої-небудь трансверсалі кубічної матриці *n*-го порядку можна співставити його графік, що являє собою ломану криву, яка, виходячи з точки на першій горизонталі (i), що відповідає значенню $i^{(\alpha)}$ індексу i цього елементу, проходить через точки наступних горизонталей (j), (k), що відповідають значенням $j^{(\alpha)}$, $k^{(\alpha)}$ індексів j, k. Побудовані таким чином графіки елементів трансверсалей взаємно перетинаються, за виключенням графіків елементів головної діагоналі $A_{111}, A_{222}, ..., A_{nnn}$, що являють собою вертикальні відрізки. Позначаючи, як і раніше, кількість інверсій в перестановках, що утворюються значеннями індексів i, j, k в елементах трансверсалі, що утворюють добуток

$$A_{i^{(1)}j^{(1)}k^{(1)}}A_{i^{(2)}j^{(2)}k^{(2)}}...A_{i^{(n)}j^{(n)}k^{(n)}},$$

відповідно через I_i, I_j, I_k, ми виходячи з розглянутої раніше діаграми робимо висновок, що число N_{ij} перетинів графіків цих елементів між горизонталями (i) і (j) буде тієї ж парності, як і сума I_i+I_j, а число N_{jk} перетинів між горизонталями (j) і (k) – тієї ж парності, як і сума I_j+I_k. Звідси випливає, що число N_{ik} перетинів між горизонтальними (i) і (k), дорівнює $N_{ij}+N_{jk}$, має парність, що співпадає з парністю суми I_i+I_k. Таким чином, перемножуючи зазначені вище добутки елементів трансверсалі на один з додаткових множників [69]

$$(-1)^{N_{ij}},\ (-1)^{N_{jk}},\ (-1)^{N_{ik}},$$

ми отримуємо відповідно член кубічного детермінанта з однією з сигнатур $(\overset{+}{k})$, $(\overset{+}{i})$, $(\overset{+}{j})$.

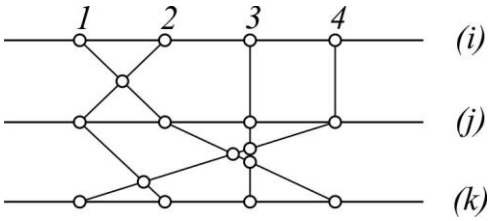


Рисунок 2.7 – Діаграма кубічної матриці 4-го порядку

Так, з діаграми (рис.2.7), побудованій по елементам трансверсалі A_{124}, A_{212}, A_{333}, A_{441} кубічної матриці 4-го порядку $\left\|A_{ijk}\right\|$ $(i, j, k = 1,2,3,4)$, бачим, що $N_{ij} = 1,\ N_{jk} = 4,\ N_{ik} = 5.$

Таблиця значень N в залежності від сигнатури детермінанта має вигляд:

Таблиця 2.1

Параметри трансверсалів кубічної матриці

Сигнатура	N	Знак додаткового множника
$(\overset{+}{i}\ \overset{\pm}{j}\ \overset{\pm}{k})$	$N_{jk} = 4$	$(-1)^{N_{jk}} = (-1)^4 = +1$
$(\overset{\pm}{i}\ \overset{+}{j}\ \overset{\pm}{k})$	$N_{ik} = 5$	$(-1)^{N_{ik}} = (-1)^5 = -1$
$(\overset{\pm}{i}\ \overset{\pm}{j}\ \overset{+}{k})$	$N_{ij} = 1$	$(-1)^{N_{ij}} = (-1)^1 = -1$

Отже, вираз $+A_{124}A_{212}A_{333}A_{441}$ є член детермінанта $\left|A_{\overset{\pm}{i}\overset{\pm}{j}\overset{\pm}{k}}\right|_4$, тоді як вираз $-A_{124}A_{212}A_{333}A_{441}$ є членом детермінантів $\left|A_{\overset{\pm}{i}\overset{+}{j}\overset{\pm}{k}}\right|_4$ і $\left|A_{\overset{\pm}{i}\overset{\pm}{j}\overset{+}{k}}\right|_4$.

Кількість m альтернативних індексів завжди передбачається парною, оскільки тоді і тільки тоді зберігається комутативність множення. Це число визначає *рід* тривимірного детермінанту: *гіпердетермінант* (коли всі індекси альтернативні), *перманент* (коли всі індекси неальтернативні), *змішаний* [69].

Детермінанти однієї і тієї ж просторової матриці об’єднуються спільною назвою *кодетермінанти*. Два кодетермінанта парного числа вимірів будуть

союзними, якщо альтернативні індекси одного є неальтернативними індексами іншого, і навпаки [68].

Детермінанти двох матриць, у яких однакова сигнатура, називаються *косигнатурними*. [69]

Узагальнюючи графічний спосіб визначення додаткового множника до добутку елементів даної трансверсалі, не трудно переконатись, що в виразі члена

$$(-1)^N A_{i_1^{(1)} i_2^{(1)} i_3^{(1)}} A_{i_1^{(2)} i_2^{(2)} i_3^{(2)}} \ldots A_{i_1^{(n)} i_2^{(n)} i_3^{(n)}}$$

тривимірний детермінант роду m з сигнатурою $(\overset{+}{i}_1 \overset{\pm}{i}_2 \overset{\pm}{i}_3)$, $(\overset{\pm}{i}_1 \overset{+}{i}_2 \overset{\pm}{i}_3)$ або $(\overset{\pm}{i}_1 \overset{\pm}{i}_2 \overset{+}{i}_3)$ показник N тривимірного детермінанта визначається формулою:

$$N = \sum_{\nu=1}^{\frac{m}{2}} N_{\alpha_{2\nu-1}\alpha_{2\nu}},$$

де $N_{\alpha_{2\nu-1}\alpha_{2\nu}}$ кількість перетинів графіків елементів, що входять в вираз даного члена, між горизонталями $(i_{\alpha_{2\nu-1}})$ і $(i_{\alpha_{2\nu}})$ відповідної діаграми.

2.3. Лінійні перетворення тривимірних даних.

Добуток матриці і числа. Множення матриць на число і суму (різницю) матриць однакових розмірів визначається так само, як і для двомірних матриць.

Добуток багатомірних матриць. Множення багатомірних матриць аналогічно множенню двомірних матриць, але більш різноманітне і виконується з використанням правил тензорного числення: *зачеплення*, *згортання* або *скорочення індексів*. Якщо деякий індекс зустрічається у виразі двічі, то вираз має бути по цьому індексу підсумовано. Наприклад:

$$a_i b_i = \sum_i a_i b_i,\ a_{ij} b_j = \sum_j a_{ij} b_j,\ a_{ij} b_i = \sum_j a_{ij} b_i$$

Тобто можна отримувати різні добутки одних і тих же матриць, змінюючи сукупність індексів.

Добуток двох тривимірних матриць. Застосуємо до трилінійної форми $F = \sum_{i,j,k=1}^{n} A_{ijk} x_i y_j z_k$ лінійне перетворення [69]

$$x_\lambda = \sum_{i=1}^{n} a_{\lambda i} X_i \qquad (\lambda = 1,2,...,n) \tag{2.30}$$

з матрицею $a = \left\| a_{\lambda i} \right\|$ $(\lambda, i = 1,2,...,n)$. В результаті отримуємо трилінійну форму $F' = \sum_{i,j,k=1}^{n} A'_{ijk} X_i y_i z_k$, де

$$A'_{ijk} = \sum_{\lambda=1}^{n} A_{\lambda jk} a_{\lambda i} \tag{2.31}$$

Так само, застосовуємо до форми F лінійне перетворення

$$y_\lambda = \sum_{j=1}^{n} a_{\lambda j} Y_j \qquad (\lambda = 1,2,...,n) \tag{2.32}$$

або

$$z_\lambda = \sum_{k=1}^{n} a_{\lambda k} X_k \qquad (\lambda = 1,2,...,n) \tag{2.33}$$

з тією ж матрицею a, як у перетворенні (2.8), отримуємо відповідну трилінійну форму $F'' = \sum_{i,j,k=1}^{n} A''_{ijk} x_i Y_j z_k$ або $F'' = \sum_{i,j,k=1}^{n} A''_{ijk} x_i y_j Z_k$, де

$$A''_{ijk} = \sum_{\lambda=1}^{n} A_{i\lambda k} a_{\lambda j}, \tag{2.34}$$

$$A'''_{ijk} = \sum_{\lambda=1}^{n} A_{ij\lambda} a_{\lambda k}. \tag{2.35}$$

Кубічну матрицю n-го порядку $A' = \left\| A'_{ijk} \right\|$ $(i, j, k = 1,2,...,n)$ форми F' називається [68] *добутком за індексом* i кубічної матриці n-го порядку

$A=\left\|A_{ijk}\right\|$ $(i,\ j,k=1,2,...,n)$ форми F на квадратну матрицю того ж порядку a лінійного перетворення цієї ж форми і позначать $A\{i\}a$.

Так само матрицю $A''=\left\|A''_{ijk}\right\|$ $(i,\ j,k=1,2,...,n)$ форми F'' або матрицю $A'''=\left\|A'''_{ijk}\right\|$ $(i,\ j,k=1,2,...,n)$ форми F''' будем відповідно називати *добутком за індексом* j *або* k матриці A на a і позначать як $A\{j\}a$ або $A\{k\}a$.

Таким чином, беручи за основу формули (2.8), (2.11), (2.12) маємо [71]:

$$A\{i\}a=\left\|\sum_{\lambda=1}^{n}A_{\lambda jk}a_{\lambda i}\right\|, \tag{2.36}$$

$$A\{j\}a=\left\|\sum_{\lambda=1}^{n}A_{i\lambda k}a_{\lambda j}\right\|, \tag{2.37}$$

$$A\{k\}a=\left\|\sum_{\lambda=1}^{n}A_{ij\lambda}a_{\lambda k}\right\|, \tag{2.38}$$

тобто добуток за індексом ν (ν – будь який з індексів i, j, k) кубічної матриці n-го порядку A на квадратну матрицю того ж порядку a є кубічною матрицею n-го порядку, у якої μ-й $(1\le\mu\le n)$ елемент будь якої строки направлення (ν) це добуток відповідної строки матриці A на μ-й стовбець матриці a. [68]

Операцію знаходження добутку $A\{\nu\}a$ будемо називати добутком за індексом ν кубічної матриці A на квадратну матрицю a. Для квадратних матриць $A=\left\|A_{ij}\right\|$ і $a=\left\|a_{ij}\right\|$ $(i,\ j=1,2,...,n)$ очевидно, $A\{j\}a=Aa$ і $A\{i\}a=a'A$, де a' – транспонована матриця відносно a.

Тепер застосуємо до трилінійної форми $F=\sum_{i_1,i_2,i_3=1}^{n}A_{i_1i_2i_3}x_{i_1}y_{i_2}z_{i_3}$ білінійне перетворення

$$x_{j_1}=\sum_{j_2,j_3=1}^{n}a_{j_1j_2j_3}X_{j_2}\xi_{j_3}\quad(j_1=1,2,...,n) \tag{2.39}$$

з кубічною матрицею

$$a = \left\| a_{j_1 j_2 j_3} \right\| \qquad (j_1, j_2, j_3 = 1,2,...,n).$$

Отримаєм в результаті чотирилінійну форму

$$F' = \sum_{k_1,...,k_4=1}^{n} A'_{k_1 k_2 k_3 k_4} X_{k_1} y_{k_2} z_{k_3} \xi_{k_4},$$

де

$$A'_{k_1 k_2 k_3 k_4} = \sum_{\lambda=1}^{n} A_{\lambda k_2 k_3} a_{\lambda k_1 k_4} \tag{2.40}$$

Так само, застосовуючи до форми F білінійне перетворення

$$y_{j_1} = \sum_{j_2, j_3=1}^{n} a_{j_1 j_2 j_3} Y_{j_2} \eta_{j_3} \quad (j_1 = 1,2,...,n) \tag{2.41}$$

або

$$z_{j_1} = \sum_{j_2, j_3=1}^{n} a_{j_1 j_2 j_3} Z_{j_2} \zeta_{j_3} \quad (j_1 = 1,2,...,n) \tag{2.42}$$

з тією ж матрицею a, як у перетворенні (2.16), ми отримуємо відповідно чотирилінійну форму [70]

$$F'' = \sum_{k_1,...,k_4=1}^{n} A''_{k_1 k_2 k_3 k_4} x_{k_1} Y_{k_2} z_{k_3} \eta_{k_4} \text{ або } F''' = \sum_{k_1,...,k_4=1}^{n} A'''_{k_1 k_2 k_3 k_4} x_{k_1} y_{k_2} Z_{k_3} \xi_{k_4},$$

де

$$A''_{k_1 k_2 k_3 k_4} = \sum_{\lambda=1}^{n} A_{k_1 \lambda k_3} a_{\lambda k_2 k_4}, \tag{2.43}$$

$$A'''_{k_1 k_2 k_3 k_4} = \sum_{\lambda=1}^{n} A_{k_1 k_2 \lambda} a_{\lambda k_3 k_4} \tag{2.44}$$

Чотиривимірна матриця *n*-го порядку

$$A' = \left\| A'_{k_1 k_2 k_3 k_4} \right\|, \; A'' = \left\| A''_{k_1 k_2 k_3 k_4} \right\|, \; A''' = \left\| A'''_{k_1 k_2 k_3 k_4} \right\|$$

$$(k_1, k_2, k_3, k_4 = 1,2,...,n)$$

форм F', F'', F''' буде тоді відповідати *добуткам*

$$A\{i_1\}a,\ A\{i_2\}a,\ A\{i_3\}a$$

за індексами i_1, i_2, i_3 кубічної матриці *n*-го порядку A на кубічну матрицю того ж порядку a. Ці добутки виходячи з формул (2.17), (2.20), (2.21) мають вигляд

$$A\{i_1\}a = \left\| \sum_{\lambda=1}^{n} A_{\lambda k_2 k_3} a_{\lambda k_1 k_4} \right\| \quad (k_1, k_2, k_3, k_4 = 1,2,...,n) \tag{2.45}$$

$$A\{i_2\}a = \left\| \sum_{\lambda=1}^{n} A_{k_1 \lambda k_3} a_{\lambda k_2 k_4} \right\| \quad (k_1, k_2, k_3, k_4 = 1,2,...,n) \tag{2.46}$$

$$A\{i_3\}a = \left\| \sum_{\lambda=1}^{n} A_{k_1 k_2 \lambda} a_{\lambda k_3 k_4} \right\| \quad (k_1, k_2, k_3, k_4 = 1,2,...,n) \tag{2.47}$$

Добуток просторових матриць не має комутативних властивостей. Складання просторових матриць і їх добуток пов'язані законом дистрибутивності.

Одинична матриця. Розглянем матричне рівняння

$$^{\lambda,\mu}(AX) = A \tag{2.48}$$

де

$$A = \left\| A_{i_1 i_2 i_3} \right\| \quad (i_1, i_2, i_3 = 1,2,...,n)$$

тривимірна матриця *n*-го порядку над полем P, а X – невідома матриця того ж порядку n.

Фактично X це одинична матриця *n*-го порядку

$$E = \left\| E_{csm} \right\| \tag{2.49}$$

в якій кожен елемент E_{csm} дорівнює δ_{cm} (символу Кронекера), а c, s, m – вже згадані вище розбиття сукупності індексів $j_1, j_2, ..., j_\tau$.

Матрицю (2.48) називають (λ, μ) – *одиничною матриці порядку* n і позначають як $E(\lambda, \mu)$ [69]. Як легко переконатись, вона перестановочна з будь якою тривимірною матрицею *n*-го порядку A:

$$^{\lambda,\mu}(AE) = ^{\lambda,\mu}(EA) = A \tag{2.50}$$

Кількість одиниць в матриці $E(\lambda,\mu)$ дорівнює $n^{\lambda+\mu}$.

$$E(\lambda,\mu) = \|E_{csm}\| = \left\| \begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array} \right\| \begin{array}{l} \rightarrow (s) \\ \downarrow \\ (c) \end{array} \quad \rightarrow (m)$$

Обернена матриця. Розглянем тепер рівняння

$$^{\lambda,\mu}(AX) = E(\lambda',\mu') \tag{2.51}$$

де $A = \|A_{isc}\|$ – та ж матриця, що і в рівнянні (2.39), а $E(\lambda',\mu')$ це (λ',μ')-одинична тривимірна матриця порядку n.

Матриця X, що влаштовує рівняння (2.42), називається (λ,μ)-обернена матриця для A відносно $E(\lambda',\mu')$ і позначається як A^{-1} або більш детально $A^{-1}(\lambda,\mu)$ [69]

Для знаходження матриці $A^{-1}(\lambda,\mu)$ маємо систему лінійних рівнянь з невідомим X_{csm}:

$$\sum_{(c)} A_{isc} X_{csm} = E_{lsm} \tag{2.52}$$

де індекси сукупності lsm утворюють сукупність індексів $c_1 ... c_\mu$, $s_1 ... s_\lambda$, $m_1 ... m_{\mu'}$ матриці $E(\lambda',\mu')$. Існуванням рішення такої системи обумовлюється існування матриці $A^{-1}(\lambda,\mu)$.

Детермінант системи рівнянь (2.43) буде тоді дорівнювати $\Delta^{n^\nu}_{\lambda,\mu'}$, де

$$\Delta_{\lambda,\mu} = \prod_{h=1}^{n^\lambda} \left| A_{ls^{(h)}c} \right| \quad (l = l^{(1)}, l^{(2)}, ..., l^{(n^\mu)}; c = c^{(1)}, c^{(2)}, ..., c^{(n^\mu)}) \tag{2.53}$$

2.4. Нелінійні перетворення тривимірних даних.

Криптографічна якість S-блоків підстановки і їх здатність протистояти атакам криптоаналізу значно поліпшується зі зростанням довжини N. Таким чином, актуальною є задача синтезу S-блоків підстановки більшої довжини, тобто, багатобайтних.

Зі зростанням довжини S-блока підстановки дуже сильно ускладнюються методи їх синтезу. Ця обставина органічно ставить задачу синтезу великих S-блоків підстановки, а також розробку рекурентних методів синтезу S-блоків підстановки.

S-блок підстановки є нелінійним перетворенням, тому що для нього в загальному випадку не виконується принцип суперпозиції, який можна сформулювати як рівність сум перетворень пари вихідних даних перетворенню суми пари вихідних даних $C = C'$

$$\begin{aligned} C &= Ta + Tb; \\ C' &= T(a+b).'' \end{aligned} \qquad (2.54)$$

де T – перетворення S-блоку підстановки.

Наприклад є таблиця підстановки (S-блок) табл 2.2:

Таблиця 2.2

Підстановка (S-блок)

$$S = \left\{ \begin{array}{lcccccccccccccccc} X = & \{0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15\} \\ & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow & \updownarrow \\ Q = & \{15 & 14 & 13 & 11 & 6 & 12 & 9 & 2 & 5 & 10 & 4 & 8 & 0 & 1 & 3 & 7\} \end{array} \right\}$$

де $X = [x_i]$, $i = \overline{1,2^k}$ – монотонно зростаюча послідовність натуральних чисел; $Q = \lfloor y_j \rfloor$, $j = \overline{1,2^k}$ – кодуюча послідовність, що побудована з елементів послідовності X.

Нехай $a = 3$, $b = 7$, тоді $Ta = 11$, а $Tb = 2$. Відповідно, виконавши складання за модулем $C = Ta + Tb = 11 + 2 = 13$. З іншого боку $a + b = 3 + 7 = 10$,

тоді $C' = T(a+b) = 4$. Очевидно, що для цього випадку $C = 13 \neq C' = 4$, що доводить факт нелінійності перетворення S-блоку підстановки.

S-блоки підстановки сучасних криптографічних алгоритмів і методи їх побудови. На сьогоднішній день відомо багато підходів к побудові Q-послідовностей. З них можна виділити чотири головні:

– Випадково обрати. Зрозуміло, що невеликі випадкові S-блоки небезпечні, але великі випадково обрані S-блоки підстановки можуть виявитись доволі якісними [70]. Наприклад, в шифрі IDEA [23] використовуються великі залежні від ключа S-блоки підстановки;

– Обрати і перевірити. Приклади такого підходу містяться в [70], де обрано підхід к конструюванню Q-послідовностей, коли вони створюються згідно з певними критеріями методом повного перебору. Такий підхід доволі результативний, але викликає труднощі навіть при довжині Q-послідовностей $N = 16$, тому що перебирання усіх $16!$ послідовностей може зайняти багато часу. Складно уявити, як цей метод можна адаптувати до S-блоків підстановки довжиною $N = 256$;

– Розробити вручну. При цьому математичний апарат використовується дуже мало: S-блоки створюються з використанням інтуїтивних прийомів;

– Розробити математично. S-блоки створюються відповідно до математичних законів, тому вони мають гарантовану надійність по відношенню к відомим атакам криптоаналізу, а також хорошими дифузійними властивостями. Такий підхід доволі поширений, наприклад його можна зустріти в [23, 70].

Опираючись на огляд сучасних методів побудови криптографічних S-блоків підстановки і критеріїв, які до них пред’являються, можна привести класифікаційну схему (рис.2.8) відомих методів побудови S-блоків підстановки, які відповідають базовим критеріям якості. Класифікація зроблена відповідно до пріоритетів розробників.

Рисунок 2.8 – Класифікаційна схема методів синтезу S-блоків підстановки

В роботі [70] показано, що якість криптографічних S-блоків підстановки значно поліпшується зі зростанням їх довжини N, що приводить до стрімкого зростання криптостійкості шифрів, в яких вони використовуються. В теперішній час в літературі мало освітлено ефективні методи синтезу S-блоків підстановки довжиною $N > 2^k = 2^8$, тобто більше ніж однобайтні. Не існує ефективних з розрахункової точки зору методів оцінки параметрів многобайтних підстановок.

Важливою проблемою є також вивчення таких критеріїв як алгебраїчний ступінь нелінійності S-блоків підстановки, афінна незалежність їх компонентних булевих функцій [70]. Ці питання залишаються невирішеними.

Дуже цікавими є питання побудови рекурентних алгоритмів конструювання S-блоків підстановки, які мають заданий рівень криптографічної стійкості. Такі алгоритми відкривають шлях до конструювання скільки завгодно великих S-блоків підстановки за допомогою відносно нескладних алгоритмів, що є дуже цінною можливістю отримати такі конструкції.

Така проблема математики і теорії передачі інформації як синтез досконалих алгебраїчних конструкцій невід'ємно пов'язано з пробмлемами теорії синтезу S-блоків підстановки. Такими перспективними конструкціями можуть бути такі двійкові і багаторівневі послідовності [70]:

1. Бент-функції, або бент-послідовності;
2. Досконалі двійкові решітки;
3. Послідовності де Брейна;
4. Конструкції полів Галуа: МЛРП, m-послідовності, коди ступеневих відрахувань, і т.п.;
5. Тори де Брейна.

Розробка нових алгоритмів синтезу вищенаведених конструкцій буде сприяти появі нових, більш ефективних методів синтезу високоякісних S-блоків підстановки.

Методи синтезу S-блоків підстановки, що відповідають строгому лавинному критерію. Однією з конструкцій, яка є регулярним методом синтезу високонелінійних S-блоків підстановки є конструкція Ніберг, що застосовується у криптоперетворенні Rijndael|AES. S-блоки підстановки, побудовані за правилами цієї конструкції, досягають практично максимального значення відстані нелінійності $N_S = 112$. Однак, з'ясовується, що вони не відповідають строгому лавинному критерію [70].

Основні результати досліджень залежності криптографчних властивостей S-блоків підстановки від вигляду обраного поліному наведені в [41].

Зазвичай S-блок підстановки має розмір кратний байту і зберігається у вигляді таблиці підстановок.

Наприклад, таблиця замін AES розміром 16x16 має вигляд [40]:

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 2.9 – S-блок AES

Наприклад, якщо на вході {68}, то шукається значення, що міститься за координатами 6 рядок і 8 стовбець. В результаті отримуємо число {45}.

Оскільки в AES блоки даних зберігаються як байти записані у матрицю розміром 4x4, то нелінійна заміна відбувається таким чином:

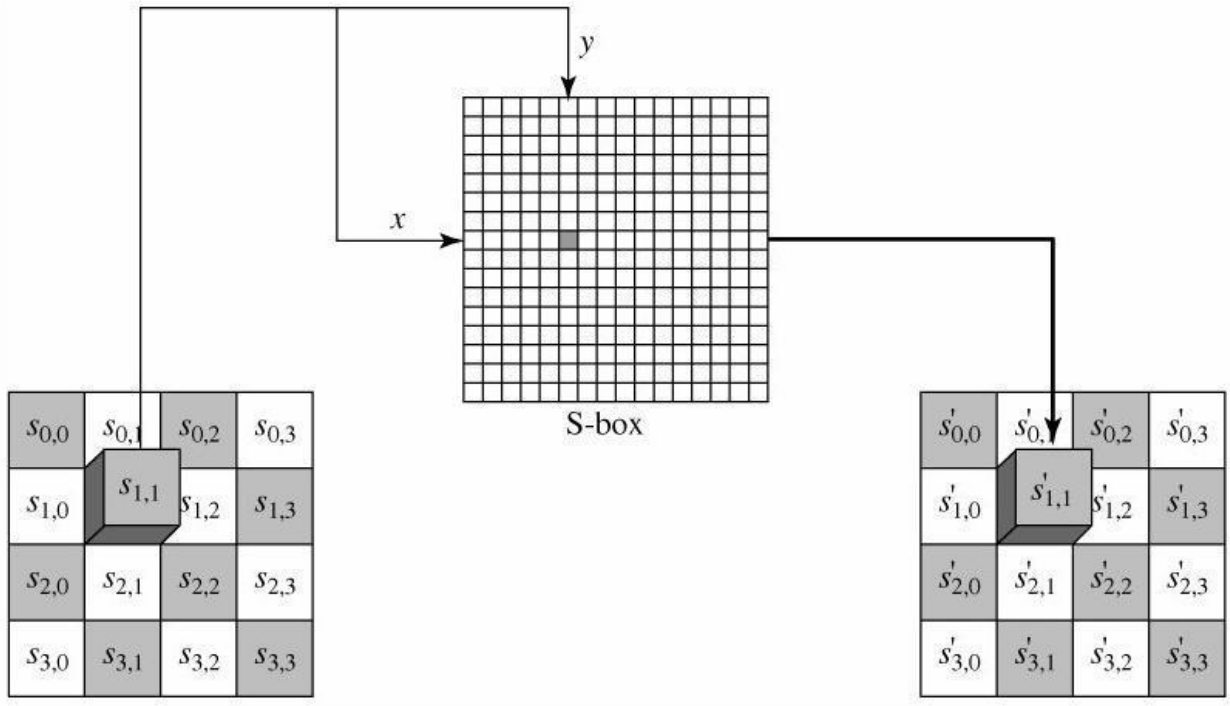


Рисунок 2.10 – Заміна байту $s_{1,1} = 68$ на байт $s'_{1,1} = 45$ в AES

При цьому використовується декілька S-блоків підстановки.

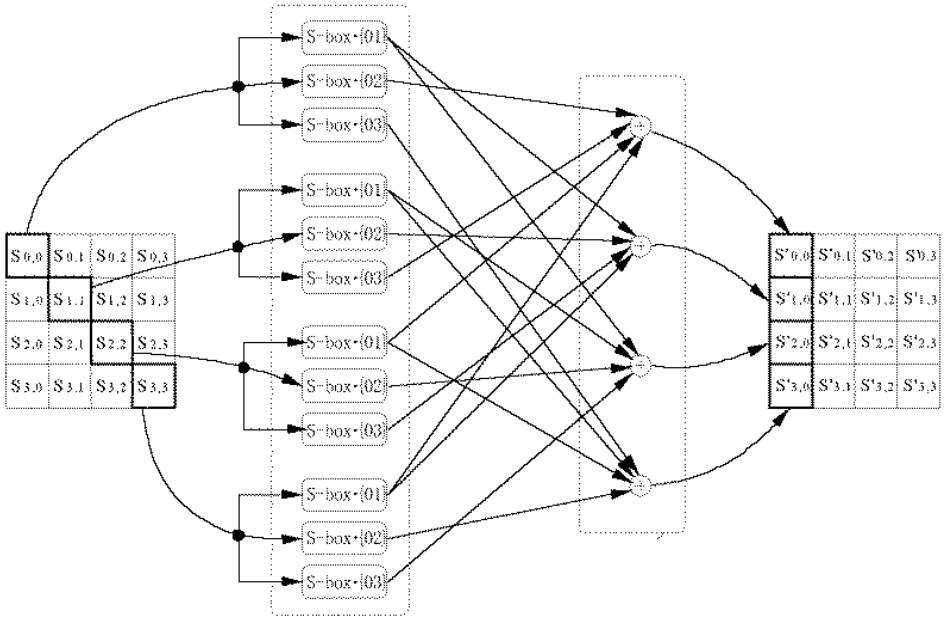


Рисунок 2.11 – Схематичне зображення використання S-box в AES

Структура S-блоку підстановки в тривимірному просторі. Як було показано вище S-блоки підстановки зазвичай мають квадратну форму або прямокутну. В тривимірному просторі вони мають кубічну форму або паралелепіпед. Слід пам’ятати, що в тривимірному просторі в залежності від обраної системи координат залежить структурування даних.

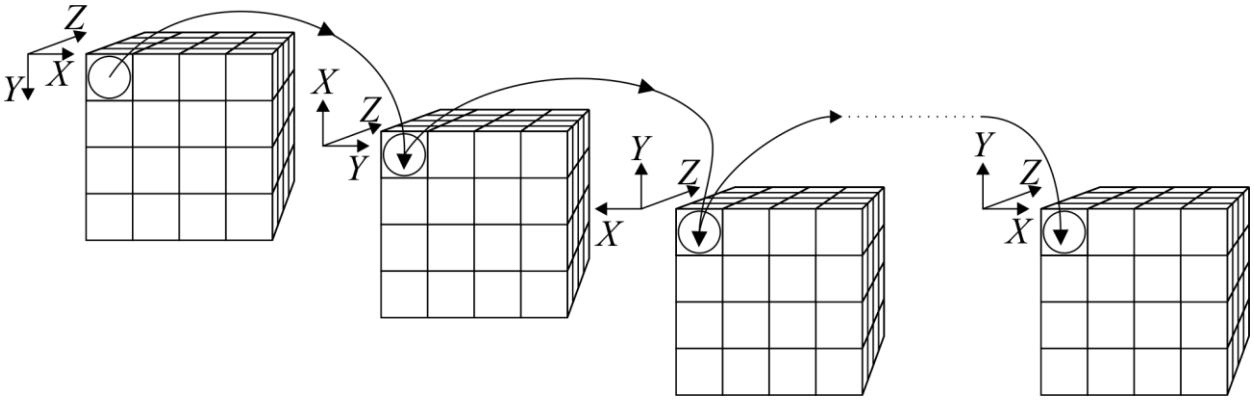


Рисунок 2.12 – Приклад зміни положення даних залежно від обраних координат

Існує декілька варіантів нелінійної підстановки в тривимірному просторі.

Взаємодія двовимірного S-блоку підстановки з перетином тривимірної матриці.

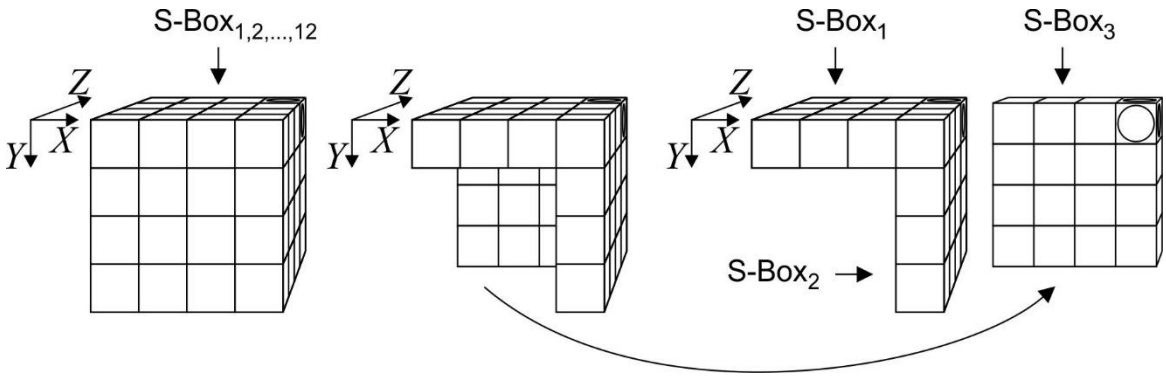


Рисунок 2.13 – Приклад впливу S-Box на перетин матриці

Перевага такого підходу у тому, що в тривимірному просторі зміна системи координат має значно більше варіацій розташування даних у просторі.

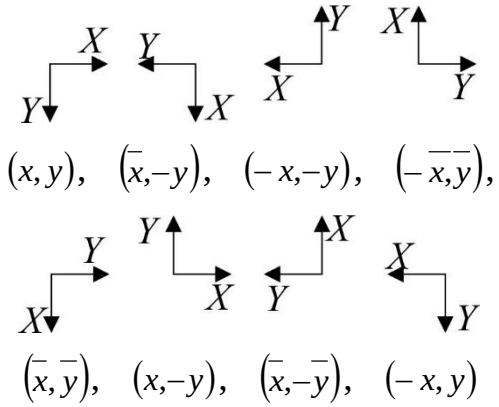


Рисунок 2.14 – Можливі варіанти координат для двовимірних матриць

Звісно, можливі інші варіанти перетворення S-Box, наприклад рандомним зсувом. Але проста заміна координат найшвидша.

Слід зазначити, що матрицю розміром 4x4 можна заповнити 16! варіантами.

Основні переваги тривимірної нелінійної підстановки:

1) Кожен елемент даних оброблюється трьома різними таблицями підстановок. Оскільки заміна іде по трьом напрямкам з використанням різних S-Box для кожного перетину тривимірної матриці;

2) Простий генератор таблиць підстановки шляхом зміни напрямку координат.

Взаємодія тривимірного S-блоку підстановки з перетином тривимірної матриці.

Суть полягає в тому, що S-блок підстановки має тривимірну форму. При цьому декілька S-блоків підстановки можуть формувати значно більший за розміром S-блок підстановки.

2.5. Метод синтезу тривимірних узагальнених матриць Галуа

Для підвищення швидкості роботи при зменшенні ресурсоємності шифратора автор пропонує метод синтезу тривимірних узагальнених матриць Галуа (УМГ) [8] для криптографічних перетворень, який грунтується на тривимірних операторах.

Будь-яка система з n^3 елементів $A_{i,j,k}$ $(i, j, k = 1,2,\ldots,n)$ поля $GF(p)$, розташованих у точках тривимірного простору, що визначається координатами i, j, k, називається *тривимірною (кубічною) матрицею n-го порядку над* полем $GF(p)$ і позначається як $\|A_{i,j,k}\|$ або $Q^{(n)}$. Сукупність елементів матриці $\|A_{i,j,k}\|$ з фіксованим значенням індекса i називається *перетином орієнтації* (i) та для простоти позначають так $A_1, A_2, \ldots, A_n$. Аналогічно визначаються перетини орієнтацій (j) і (k).

Нехай $\circ$ це деякий оператор, завдяки якому перетини $A_1, A_2, \ldots, A_n$ утворюють кубічну матрицю

$$Q^{(n)} = A_1 \circ A_2 \circ \cdots \circ A_n,$$

що дозволяє формально ввести правила:

$$\overline{Q}^{(n)} = \overline{A}_n \circ \overline{A}_{n-1} \circ \cdots \circ \overline{A}_2 \circ \overline{A}_1.$$

$$\det\left(Q^{(n)}\right) = \det(A_1) \circ \det(A_2) \circ \cdots \circ \det(A_n).$$

Як оператор $\circ$ може бути обраний оператор модульного множення $\otimes$, конкатенації $\|$ тощо.

Завдяки певним криптографічним властивостям, як матриці $A_1, A_2, \ldots, A_n$ обрані так звані узагальнені примітивні матриці Галуа $G_{f,\theta}^{(n)}$. Термін «матриця Галуа» запозичений з теорії кодування і криптографії, у якій широко використовуються генератори ПВП в конфігурації Галуа, що засновані на регістрах зсуву з лінійними зворотними зв'язками (РЗЛЗЗ). Відомо, що для того, щоб РЗЛЗЗ був генератором максимального періоду, відповідний поліном зворотного зв'язку повинен бути примітивним поліномом. Кожен лінійний генератор ПВП на РЗЛЗЗ може бути представлений відповідною матрицею Галуа, що формує ту саму послідовність, що і генератор ПВП.

Синтез примітивних узагальнених матриць Галуа n-го порядку відбувається за методом діагонального заповнення, суть якого полягає в наступному: у нижньому рядку матриці записується примітивний елемент поля $GF\left(2^n\right)$, що породжується незвідним поліномом f_n, який не обов'язково має бути примітивним. Наступні рядки матриці утворюються зсувом попереднього рядка на один розряд ліворуч. Якщо при цьому старший ненульовий розряд рядка виходить за межі матриці, то поліноми, що відповідають таким рядкам, приводяться до залишку за модулем f_n і рядок знову стане n розрядним.

Із теорії поліномів однієї змінної x відомо, що множення довільного полінома $\omega_k(x)$ ступеня k на x еквівалентно його зсуву на один розряд ліворуч і, відповідно, збільшення на один ступінь полінома. Або, іншими словами,

$$x \cdot \omega_k(x) = \omega_{k+1}(x),$$

що дає можливість виконати такі перетворення над $G_{f,\omega}^{(n)}$

$$G_{f,\omega}^{(n)} \Rightarrow \begin{pmatrix} x^{n-1}\cdot\omega \\ x^{n-2}\cdot\omega \\ \vdots \\ x\cdot\omega \\ \omega \end{pmatrix} \bmod f_n = \omega \cdot \begin{pmatrix} x^{n-1} \\ x^{n-2} \\ \vdots \\ x \\ 1 \end{pmatrix} \bmod f_n = \omega \cdot$$

Твердження. *Узагальнені матриці Галуа* $G_{f,\omega}^{(n)}$ *порядку* n *над незвідними поліномами* f_n *ступеня* n *з коефіцієнтами* $a_i \in GF(p),\ i=\overline{0,n}$ *ізоморфні їх утворюючим елементам* ω*, які належать полю* $GF\left(p^n\right)$ *довільної характеристики* p*, тобто* $G_{f,\omega}^{(n)} \cong \omega$.

Таким чином, між $G_{f,\omega}^{(n)}$ і їх утворюючими елементами ω (зовсім не обов'язково примітивними) існує взаємно однозначна відповідність.

Наслідок 1. Узагальнені матриці Галуа $G_{f,\omega}^{(n,2)}$ невироджені за будь-яких параметрів f_n і ω, оскільки утворені лінійно незалежними рядками (стовбцями) матриць.

Наслідок 2. Для того щоб піднести матрицю $G_{f,\omega}^{(n,2)}$ до ступеня k, достатньо вирахувати утворюючий елемент $\omega_k = \omega^k \left(\mathrm{mod}\ f_n\right)$ і за методом діагонального заповнення скласти матрицю $G_{f,\omega_k}^{(n,2)}$ за модулем f_n.

Наслідок 3. Мінімальне ненульове значення ступеня e, яке забезпечує рівність $G_{f,\omega}^{e} = E$, збігається з порядком *ord* елемента ω, утворюючого матрицю $G_{f,\omega}^{(n,2)}$.

Наслідок 4. Матриця Галуа $G_{f,\omega}^{(n,2)}$ примітивна, якщо примітивним є утворюючий її елемент ω, тобто якщо $\omega = \theta$.

Наслідок 5. Матриці Галуа $G_{f,\omega_1}^{(n,2)}$ і $G_{f,\omega_2}^{(n,2)}$, $\omega_1 \neq \omega_2$ комутативні, оскільки є елементами однієї і тієї самої мільтиплікативної групи GF^* максимального порядку, складеної зі ступеней матриці $G_{f,\theta}^{(n,2)}$, довільний примітивний утворюючий елемент якої θ належить полю $GF\left(p^n\right)$, породженому НП f_n.

Наслідок 6. Довільні алгебричні перетворення (підсумовування, віднімання, множення і ділення) над матрицею Галуа або сукупністю матриць

Галуа ізоморфні таким самим перетворенням над утворюючими елементами цих матриць.

Наслідок 7. Добуток матриці Галуа $G_{f,\omega}^{(n,2)}$ на вектор n-го порядку $\overline{V}$ збігається з добутком цього вектора на утворюючий елемент ω матриці G, тобто $G_{f,\omega}^{(n,2)} \cdot \overline{V} = \left(\overline{V} \cdot \omega\right) \bmod f_n$.

Подібні матриці Галуа можуть бути використані для реалізації протоколу обміну секретними ключами по відкритих каналах зв'язку

$$^{*}G_{f,\omega}^{(n,2)} = P^{-1} \cdot G_{f,\omega}^{(n,2)} \cdot P,$$

де $P-$ матриця подібності.

Як матрицю подібності можа прийняти будь-яку невироджену матрицю. Для простоти використовується переставна (комутативна) матриця.

На відміну від початкових УМГ $G_{f,\omega}^{(n,2)}$ подібні матриці $^{*}G_{f,\omega}^{(n,2)}$, залишаючись комутативними, втрачають властивість ізоморфізму. Ця особливість подібних матриць Галуа забезпечує можливість побудови односторонніх функцій, які використовуються в запропонованих далі протоколах обміну секретними ключами шифрування.

Протоколом передбачається формування абонентами мережі односторонньої функції, за допомогою якої і обчислюється загальний секретний ключ шифрування.

Як відкритий ключ протоколу прийняті: вектор ініціалізації *V*, який є *n*-бітовим вектором; незвідний двійковий поліном f_n ступеня *n* і перестановки *P*-матриця *n*-го порядку. Кожен з абонентів мережі *A* і *B* виробляє секретні *n* -бітові ключі ω_α і ω_β відповідно. Загальний секретний ключ *K* визначається в результаті виконання абонентами таких двох етапів обчислень:

Етап 1. Абонент *A* генерує випадковий вектор ω_α, знаходить спочатку ОМГ $G_{f,\omega_\alpha}^{(n)}$, потім подібну матрицю $^{*}G_{f,\omega_\alpha}^{(n)}$, обчислює вектор $V_\alpha = V \cdot G_{f,\omega_\alpha}^{(n)}$ і напрямляє його абоненту *B*. Аналогічні операції здійснює абонент *B*, визначаючи вектор

$V_\beta = V \cdot G^{(n)}_{f,\omega_\beta}$, який напрямляє абоненту *A*. Вектори V_α і V_β саме і є тими односторонніми функціями φ, які побудовані на основі подібних ОМГ.

Етап 2. Абонент *A* примножує отриманий від абонента *B* вектор V_β на свою секретну матрицю ${}^*G^{(n)}_{f,\omega_\alpha}$, формуючи ключ

$$K_\alpha = V_\beta \cdot {}^*G^{(n)}_{f,\omega_\alpha} = V \cdot {}^*G^{(n)}_{f,\omega_\beta} \cdot {}^*G^{(n)}_{f,\omega_\alpha} = V \cdot (P^{-1} \cdot G^{(n)}_{f,\omega_\beta} \cdot P) \cdot (P^{-1} \cdot G^{(n)}_{f,\omega_\alpha} \cdot P) = V \cdot \left(P^{-1} \cdot G^{(n)}_{f,\omega_\beta} \cdot G^{(n)}_{f,\omega_\alpha} \cdot P\right) \quad (2.55)$$

Такі самі ж обчислення виконує абонент B, розраховуючи вектор

$$K_\beta = V \cdot \left(P^{-1} \cdot G^{(n)}_{f,\omega_\alpha} \cdot G^{(n)}_{f,\omega_\beta} \cdot P\right). \quad (2.56)$$

Оскільки ОМГ $G^{(n)}_{f,\omega_\alpha}$ і $G^{(n)}_{f,\omega_\beta}$ комутативні, зі співвідношень (2.55) і (2.56) випливає, що $K_\alpha = K_\beta$ і, отже, обидва абоненти мережі отримують однаковий секретний ключ шифрування *K*.

Якщо ж замість подібних матриць ${}^*G^{(n)}_{f,\omega}$ використовувати звичайні ОМГ $G^{(n)}_{f,\omega}$, то через їх ізоморфізм супротивник, перехопивши вектори V_α і V_β, може обчислити секретні ключі ω_α і ω_β, оскільки в загальному випадку

$$V_\gamma = V \cdot G^{(n)}_{f,\omega_\gamma} = V \cdot \omega_\gamma \geq (\mathrm{mod}\, f_n), \ \gamma = \alpha \text{ або } \beta. \quad (2.57)$$

З огляду на те, що *V* і f_n – відомі величини, противник, розраховує рівність (2.57) щодо ω_γ, обчислює ключі ω_α і ω_β, це призводить до зламу загального секретного ключа *K*.

2.6. Висновки до другого розділу

В тривимірній матриці кожен перетин це двовимірна матриця. Якщо виконувати якісь перестановки в одній площині, а потім в іншій (XY, YZ, XZ), то елементи матриць з однієї двомірної матриці будуть потрапляти в іншу. І якщо після таких дій перетворити тривимірну матрицю даних в одновимірний масив даних, то отримуємо ще більш спотворену послідовність (ніж при перетворенні двовимірної структури у одновимірну).

Прості перетворення у тривимірному просторі приводять до більших спотворень у одновимірній інформаційній послідовності, ніж двовимірні. Щоб

досягти таких спотворень використовуючи одновимірну структуру треба суттєво ускладнювати алгоритм. А в тривимірному просторі використання простих операцій повороту і віддзеркалення дають результати співставні зі складними одновимірними алгоритмами.

Розробленно метод формування динамічно керованих примітивів лінійного розсіювання, нелінійній заміні та «ковзного кодуванню» на основі узагальнених перетворень Грея та матриць Галуа для тривимірного простору забезпечує підвищення криптографічної стійкості блокових шифрів порівняно з використанням одно і двовимірного представлення даних.

Запропонований протокол формування секретних ключів (ключова угода) на відміну від відомих протоколів використовує як основне перетворення однобічну функцію, що забезпечує спрощення обчислень.

Основні наукові результати отримані в даному розділі опубліковані у працях автора [2,3,5,7,8].

РОЗДІЛ 3
СИНТЕЗ І АНАЛІЗ 3D КРИПТОГРАФІЧНИХ ПРИМІТИВІВ

Криптографічні примітиви синтезовані на базі лінійних і нелінійних тривимірних перетворень. До лінійних відносяться: перемішування, ковзне кодування, добуток тривимірних матриць. До нелінійних відносяться: нелінійна заміна з використанням SBox3D.

3.1. Криптографічний примітив «Permutations»

Будь-яке перетворення даних починається з їх представлення в певному вигляді. Наприклад, файл зазвичай розглядається як одновимірний масив байт. Різні потоки даних (звук, відео, тощо) зазвичай теж розглядаються як одновимірні масиви. При криптографічному перетворенні в блочній криптографії присутня практика умовного розділення даних на блоки певного розміру (див.рис.3.1). Взагалі, блоки можна записувати в будь якому вигляді, це може бути 1D, 2D, 3D і навіть ND вимірне представлення даних. Тобто від одновимірного масиву даних до матриць, а потім до просторових матриць і аж до гіперпросторового представлення.

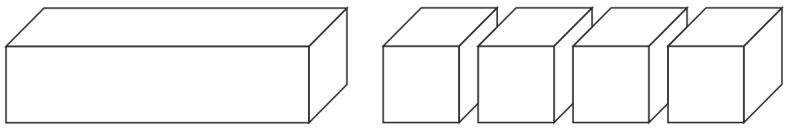


Рисунок 3.1 – Розділення даних на блоки

Виконуючи перемішування в 1D просторі ми маємо тільки одну вісь за якою можемо виконувати переміщення даних. Звісно, маючи досить складну формулу що описує «випадкове» перемішування можна отримати пристойні результати. Але, відслідкувати переміщення в одномірному просторі досить легко.

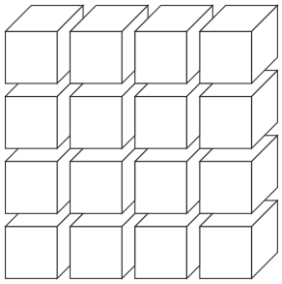


Рисунок 3.2 – Двовимірне представлення даних

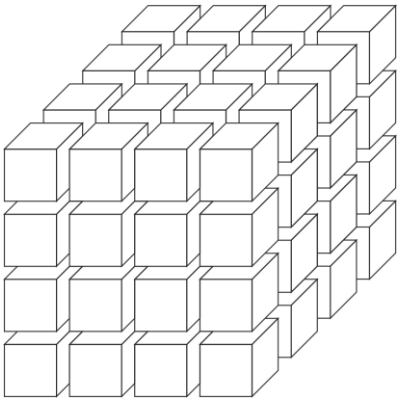


Рисунок 3.3 – Тривимірне представлення даних

В 1D мали два напрямки для переміщення даніх «право» і «вліво». В 2D маємо вже вісім напрямки переміщення: по два напрямки для горизонталі і вертикалі, і по два напрямки для діагонального переміщення. Для 3D цих напрямків вже шість для осей плюс ще вісім для чотирьох діагоналей, всього чотирнадцять варіантів переміщення. Зразу помітно, що приріст від переходу 1D до 2D складає шість ($8-2=6$), а приріст від 2D до 3D складає теж шість ($14-8=6$). Фактично це означає що перевага 2D над 1D така сама як 3D над 2D. Перевага полягає в тому, що прості (атомарні) переміщення в 2D описуються вже досить складною формулою (система рівнянь) при розгляданні в 1D. Так само будь яке незначне переміщення в 3D значно складніше описати розглядаючи його через простори 2D чи 1D. Ця обставина принципово полегшує розрахунки при виконанні операції перемішування. Оскільки обрахувати переміщення даних в просторі 3D при шифруванні (знаючи секретний ключ, що визначає порядок переміщення) не потребує багато

машинного часу. В той час як кріптоаналітику доведеться обраховувати значно більшу кількість можливостей.

Permut-64-3D-XYZ. Свідоцтво про реєстрацію авторського права на твір №48523 від 01.04.2013, комп'ютерна програма "Permut".

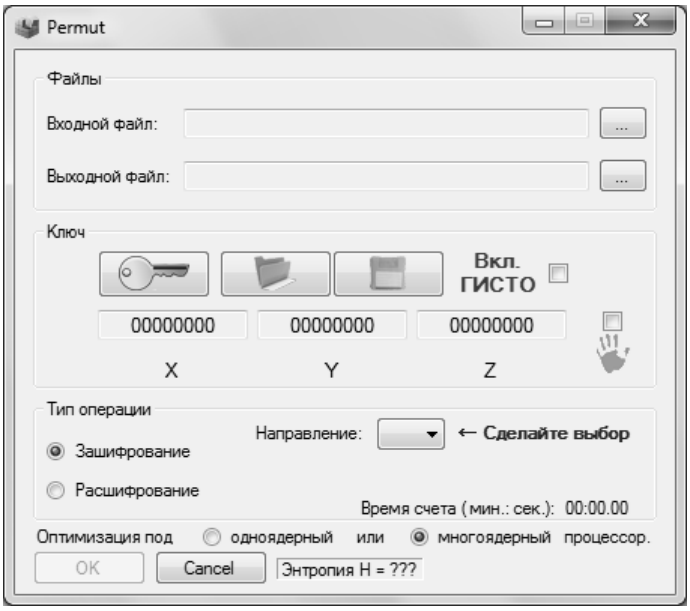


Рисунок 3.4 – Інтерфейс програми «Permut»

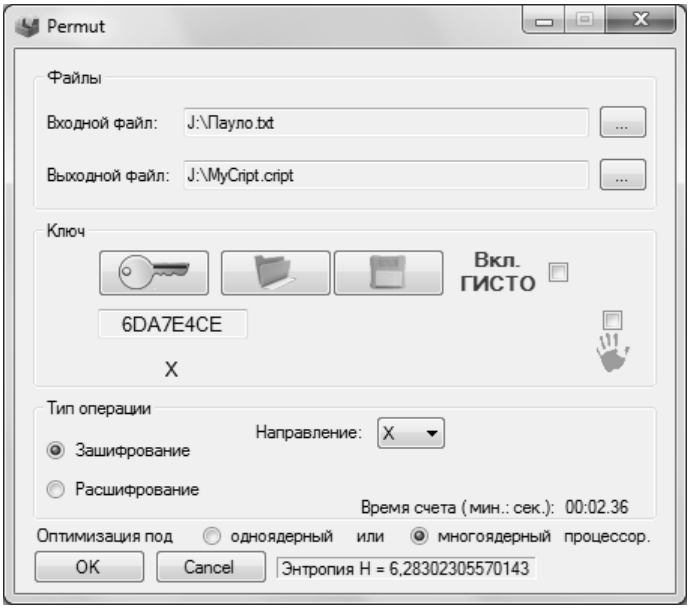


Рисунок 3.5 – Приклад роботи програми «Permut»

Програма виконує перемішування і упорядкування файлу даних, використовуючи алгоритм перемішування біт в файлі за допомогою здвигу біт в півбайтах. Біти файлу записуються в блоки (трьохвимірні матриці). Розмір

блоку 4x4x4 біта. Чотири біта, це один півбайт. Загальна кількість біт у блоку 64, загальна кількість півбайт у блоку 16. Шифрування відбувається циклічним зсувом біт в середині півбайт. Параметри зсуву приймають значення 0, 1, 2, або 3. Для цього досить 2-х бітного ключа, що приймає значення у двійковому вигляди 00, 01, 10, 11. Ключ X керує циклічним зсувом в півбайтах, колінеарних осі X, ключ Y – зсувом в півбайтах, колінеарних осі Y, а Z - колінеарних осі Z. На рис.3.6. зображено блок, в якому виділені сірим кольором півбайти колінеарні осі X. Ліворуч зображені всі 16-ть півбайт колінеарних осі X.

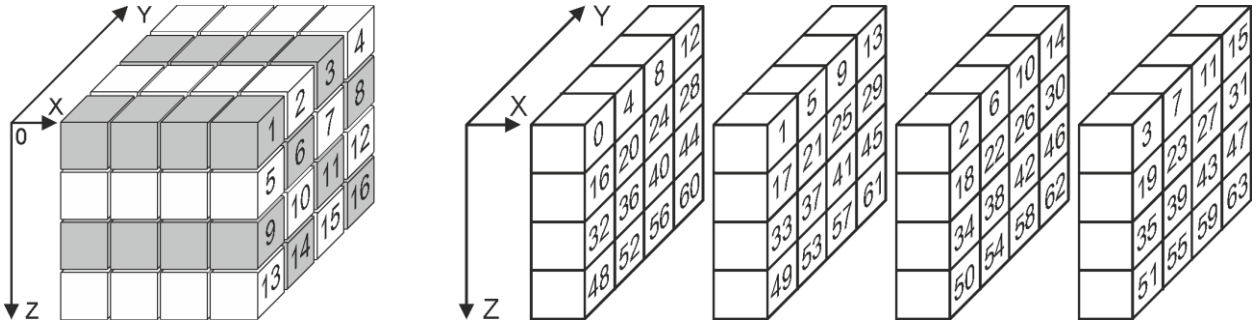


Рисунок 3.6 – Блок розміром 4x4x4 біта. Ліворуч зображені півбайти колініарні осі X. Праворуч зображено порядок заповнення блоку бітами

Ключ для кожного з напрямків X, Y, Z довжиною 8 шістнадцятирічних чисел. Кожне шістнадцятирічне число записується 4 бітами або складає два 2-х бітні ключі. Наприклад, ключ у шістнадцятирічному вигляді «31A6705D» у двійковому вигляді «0011 0001 1010 0110 0111 0000 0101 1101», де кожна пара біт відображає кількість зсувів «0301221213001131». Якщо є послідовність біт (вказуємо індекси біт) 1234 і зсув «1», то після зсуву маємо послідовність 4123. Якщо зсув «2», то 3412. Якщо зсув «3», то 2341. Алгоритм перемішування виконується в межах кожного блоку.

Permut-64-3D-123. Генератор випадкових чисел формує ключ довжиною 256 біт, наприклад «F7 A5 90 C6 08 E3 13 6C». Перші 128 біт «F7 A5 90 C6», це ключ для напрямку X. Другі 128 біт «08 E3 13 6C» для ключа напрямку Y.

Ключ для напрямку Z формується наступним чином:

1) Спочатку 256 бітний ключ, зображений у шістнадцятирічному вигляді «F7 A5 90 C6 08 E3 13 6C», записується у вигляді масиву байт. Для цього попарно беруться шістнадцятирічні числа і записуються у десятинному вигляді. Отримуємо масив з 8 байт {247, 165, 144, 198, 8, 227, 19, 108}.
2) Так як 1 байт, це 8 біт, то перетворюється масив довжиною 8 байт у масив довжиною 64 біт. Формуємо з одновимірного масиву біт тривимірний масив біт. Розмір тривимірного масиву 4x4x4 біта. Запис в тривимірний масив відбувається по координатній сітці (y, z, x).
3) Записуємо всі значення тривимірного масиву біт, використовуючи іншу координатну сітку (x, y, z), в новий одновимірний масив біт,
4) Перетворюємо одновимірний масив біт, довжиною 64 біта у одновимірний масив байт довжиною 8 байт «153, 65, 27, 49, 101, 139, 208, 126».
5) Беремо перші 4 байта масиву «153, 65, 27, 49» і створюємо з них ключ для напрямку Z «99 41 1B 31» у шістнадцятирічному вигляді.

Використовуючи алгоритм **Permut-64-3D-XYZ**, послідовно виконуємо перемішування за напрямком X, потім Y і Z (рис.3.7). Тут сірим кольором зображена колініарність до напрямків координат X, Y, Z, а цифрами від 1 до 16 зображено послідовність півбайт, в яких відбувається перемішування

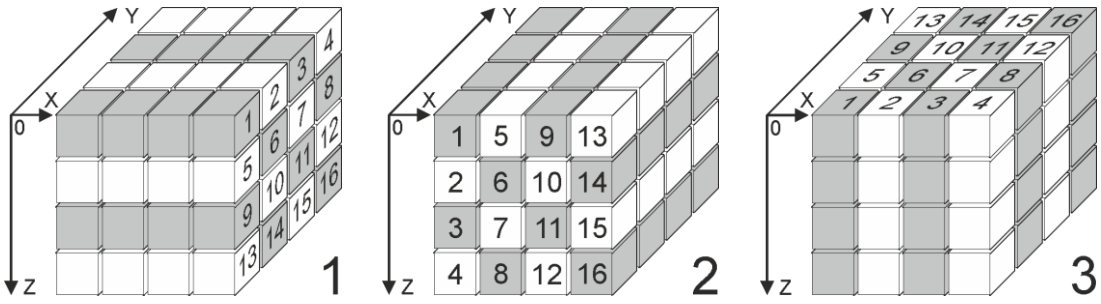


Рисунок 3.7 – Блоки розміром 4x4x4 біта.

3.2. Криптографічний примітив «Slide Coding»

Автор пропонує нове використання в 3D підкласу зворотніх примітивів, названих примітивами «ковзного кодування» (ПКК). Схема побудови ПКК [67] подібна схемам ліво- і правостороннього перетворення Грея [67] «навпаки» в тому розумінні, що прямому ковзному кодуванню відповідають схеми зворотнього перетворення Грея, а зворотному кодуванню – схеми прямих перетворень Грея. ПКК можна будувати на основі арифметичних, логічних або змішаних (арифметико-логічних) операцій перетворення елементів (сукупності *n*-бітних комбінацій) шифрованого тексту.

Перетворення (шифрування) ПКК виконується над блоками даних, що являють собою деяку сукупність байтів [67]. Зручно використовувати 32-бітний розмір елементів шифруємого тексту. Тим самим ми орієнтуємось на реалізацію алгоритмів шифрування на комп'ютерних платформах з 32-розрядними шинами. З тим же успіхом можна було б використовувати, наприклад, 64-бітний розмір шифруємих даних і шин, що не спричинить яких небудь принципових труднощів. Будемо також вважати, що шифруємоий текст поділений на блоки, кожний з яких містить парну кількість 32-бітних елементів. Остання умова означає, що розмір блоку кратний 64 бітам, тобто може бути орієнтований на обробку процесорами з 64-розрядною шиною, використання яких останнім часом стає все більш поширеним в міжнародній практиці.

Арифметичне «ковзне кодування». За аналогією з ліво- і правостороннім перетворенням Грея [3] введемо ліво- і правостороннє «ковзне кодування» з арифметичним перетворенням елементів шифрування. Для простоти будем називати його *арифметичним «ковзним кодування»* АКК. Структурна схема чотирьохелементного прямого лівостороннього АКК наведена на рис.3.8. На цьому рисунку прийняті такі позначення: $\boxplus$ - оператор арифметичного складання за mod 2^{32}; $R`$ - 32-бітний вхідний раундовий ключ; R'' – 32-бітний вихідний раундовий ключ, що використовується в якості вхідного для наступного перетворюємого блоку.

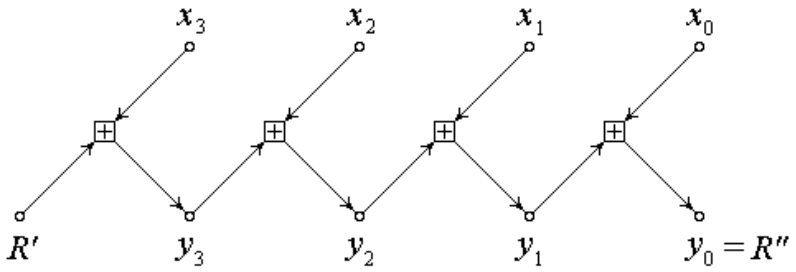


Рисунок 3.8 – Схема формування прямого лівостороннього АКК

Алгоритму прямого лівостороннього АКК відповідає система лінійних модульних алгебраїчних рівнянь

$$\begin{aligned} y_3 &= (x_3 + R')\bmod m;\\ y_2 &= (x_2 + y_3)\bmod m;\\ y_1 &= (x_1 + y_2)\bmod m;\\ y_0 &= (x_0 + y_1)\bmod m, \end{aligned} \qquad (3.1)$$

де $m = 2^{32}$.

Вирішуючи формально систему рівнянь (3.1) відносно вхідних операцій x_i, отримуємо систему, якій відповідає (рис.3.9) структурна схема чотирьохелементного зворотного лівостороннього АКК.

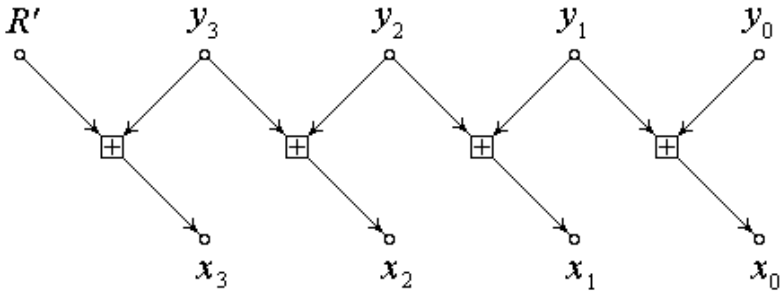


Рисунок 3.9 – Схема формування зворотного лівостороннього АКК

Приклади побудови правосторонніх АКК показані на рис. 3.10 і 3.11.

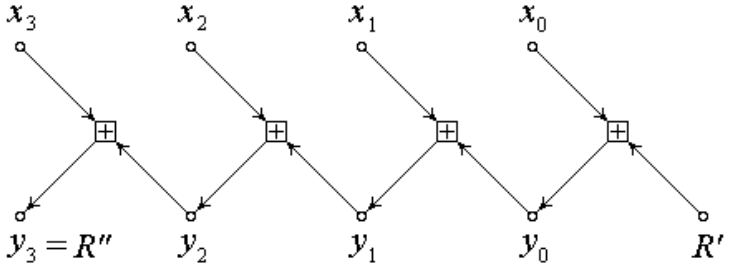


Рисунок 3.10 – Схема формування прямого правостороннього АКК

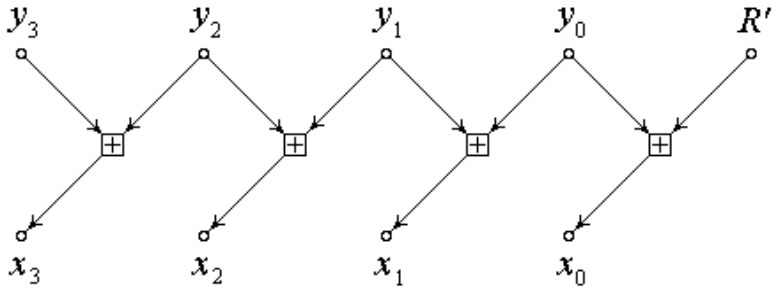


Рисунок 3.11 – Схема формування зворотного правостороннього АКК

Із співставлення схем лівостороннього (рис.3.8 і 3.9) і правостороннього (рис. 3.10 і 3.11) АКК випливає, що до схем правосторонніх АКК ми приходимо в результаті повороту на 180^{O} відносно центральній вертикальній осі схем лівосторонніх АКК при збереженні незмінними позиції операндів перетворення x і y.

Логічне «ковзне кодування». До такого виду кодування (ЛКК) приходимо в результаті логічних перетворень елементів блоків, в якості яких обрана операція порозрядного додавання за $\mathrm{mod}2$. Схеми ЛКК отримуються заміною оператора арифметичного додавання за модулем $m = 2^{32}$, який позначається $\boxed{+}$, на оператор $\oplus$ порозрядного додавання за $\mathrm{mod}2$.

В результаті вказаних замін на рис.3.8 отримуємо такий вигляд схеми прямого лівостороннього ЛКК (рис.3.12).

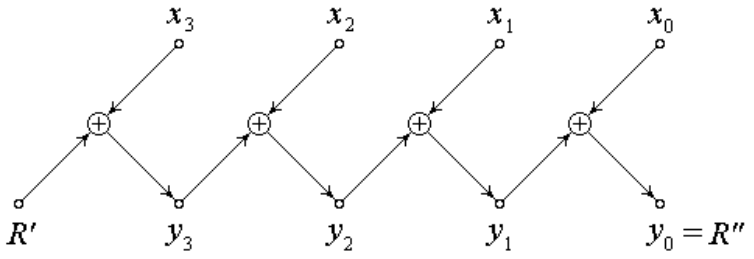


Рисунок 3.12 – Схема формування прямого лівостороннього ЛКК

Схемі перетворення, наведеній на рис.3.12, відповідає система лінійних модульних рівнянь

$$\begin{aligned} y_3 &= x_3 \oplus R'; \\ y_2 &= x_2 \oplus y_3; \\ y_1 &= x_1 \oplus y_2; \\ y_0 &= x_0 \oplus y_1, \end{aligned} \tag{3.2}$$

при цьому не слід забувати, що в даному випадку операції в (3.2) виконуються порозрядно над кожною парою суміжних операндів.

Можем наступним чином показати схему зворотного лівостороннього ЛКК (рис.3.13).

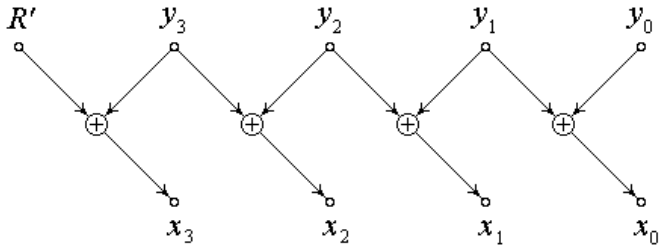


Рисунок 3.13 – Схема формування зворотного лівостороннього ЛКК

Аналогічним чином можуть бути побудовані схеми і системи лінійних модульних алгебраїчних рівнянь для прямого і зворотного правостороннього ЛКК. Звернем увагу на те, що запропоновані схеми ліво- і правостороннього логічного ковзного кодування і відповідні їм системи рівнянь є нічим іншим, як відповідні (за напрямками) *зворотні перетворення Грея* над двійковими кодовими комбінаціями, тоді як зворотні ЛКК представляють собою *прямі перетворення Грея*. Тим самим ми маємо право визначати ЛКК як перетворення Грея «навпаки».

Шифрування даних примітивами «ковзного кодування». З метою спрощення, як записів алгебраїчних перетворень, так і схем шифрування, обмежимось мінімальними розмірами тексту і мінімальними значеннями параметрів шифрування.

Будемо вважати, що відкритий текст складається з двох блоків, кожен з яких містить по два 32-бітних елемента. Позначимо через x_3, x_2, x_1 і x_0 елементи вхідного тексту, при цьому x_3 і x_2 відносяться до першого (розташованого зліва), а x_1 і x_0 - до другого блоку. Обираємо схему змішаного ковзного кодування (ЗКК), припускаючи, що на етапі зашифрування першим виконується логічне ковзне кодування (КК), а другим – арифметичне КК як для ліво-, так і правостороннього перетворення. І, нарешті, будемо вважати, що спільний ключ утворюється конкатенацією двох 32-бітних раундових ключів R_1 і R_2, а процес зашифрування, як і розшифрування, завершується за два кроки.

Схема алгоритму зашифрування на першому кроці перетворення з використанням лівостороннє ЛКК показана на рис.3.14.

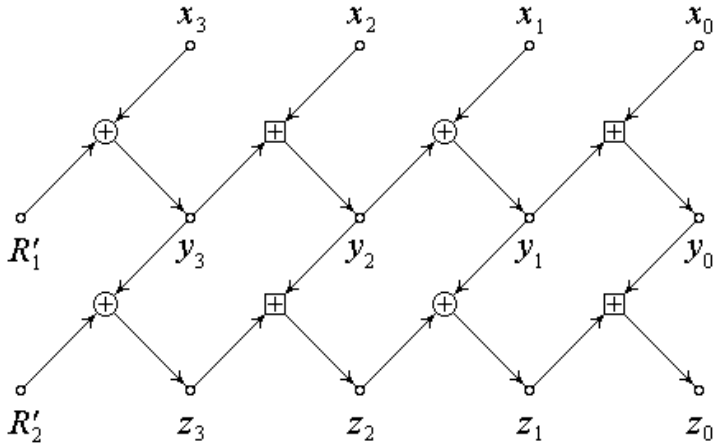


Рисунок 3.14 – Схема зашифрування ЗКК на першому раунді

Згідно з рис.3.14 перший крок зашифрування починається перетворенням (прямим лівостороннім ЗКК) операндів x під керуванням раундового ключа $R_1^{'} = R_1$. Результатом перетворення є чотирьохелементний вектор y. Завершується перший крок зишифрування перетворенням (також прямим лівостороннім ЗКК), але тепер вже операндів y під керуванням раундового ключа $R_2^{'} = R_2$. Фінальним результатом першого кроку зашифрування є вектор z.

Перед другим кроком зашифрування раундові ключі модифікуються в результаті циклічного зсуву спільного ключа шифрування на сім розрядів вліво, тобто

$$R_1 \parallel R_2 <<< 7 \Rightarrow R_1'' \parallel R_2'',$$

На другому (парному) кроці зашифрування продовжується перетворення (але тепер вже в режимі прямого правостороннього ЗКК) компонента вектора z спочатку під керуванням раундового ключа R_1'', а потім ключа R_2''. Схема зашифрування на другому раунді представлена на рис.3.15.

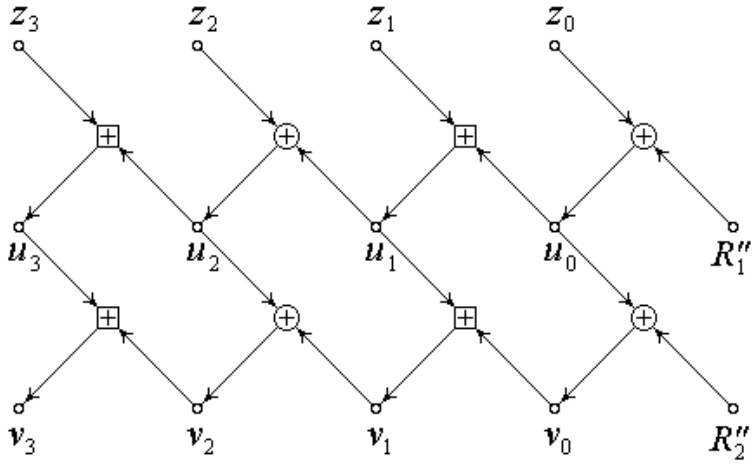


Рисунок 3.15 – Схема зашифрування ЗКК на другому раунді

Таким чином, в результаті зашифрування вхідного вектора $x = \{x_3, x_2, x_1, x_0\}$ за два кроки перетворення за схемою ЗКК під керуванням двох раундових ключів R_1 і R_2 сформовано шифротекст – чотирьохелементний вектор $v = \{v_3, v_2, v_1, v_0\}$.

Перейдемо до конструювання алгориму розшифрування криптограми з метою відновлення вхідного вектору x. Візьмемо до уваги, що зашифрування закінчено на парному кроці під керуванням раундового ключа R_2''. Тому процес розшифрування ми повинні почати з перетворення типу зворотного правостороннього ЗКК також під керуванням ключа R_2''. При цьому раундовий

ключ $R_2^{''}$, згідно з перетворенням (3.2), слід піддати циклічному зсуву на сім розрядів вліво і обрати праву половину модифікованого спільного ключа.

Скориставшись позначеннями, прийнятими на рис.3.15, запишемо систему лінійних модульних рівнянь формування елементів вектору v із елементів вектору u. Маємо:

$$\begin{aligned} v_0 &= u_0 \oplus R_2^{''}; \\ v_1 &= (u_1 + v_0) \bmod m; \\ v_2 &= u_2 \oplus v_1; \\ v_3 &= (u_3 + v_2) \bmod m. \end{aligned} \tag{3.3}$$

Система модульних рівнянь (3.3), будучи продовжена на відновлення вектору z, приводить до схеми (рис.3.16) першого кроку розшифрування (нагадуємо, що він є перетворенням, зворотним зашифруванню на парному кроці, тобто зворотнім правостороннім ЗКК).

Спираючись на наведену вище методику, можна побудувати як схеми, так і системи лінійних модульних рівнянь, що відповідають алгоритмам ковзного кодування (криптоперетворення) з заданими параметрами шифрування.

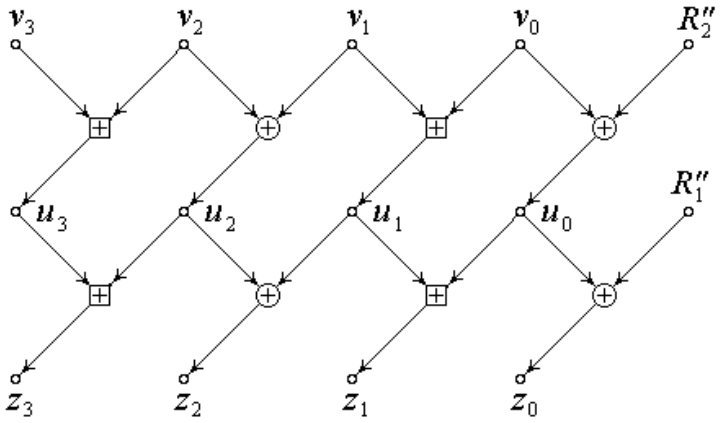


Рисунок 3.16 – Схема розшифрування ЗКК на першому раунді

Схема фінального розшифрування показана на рис.3.17.

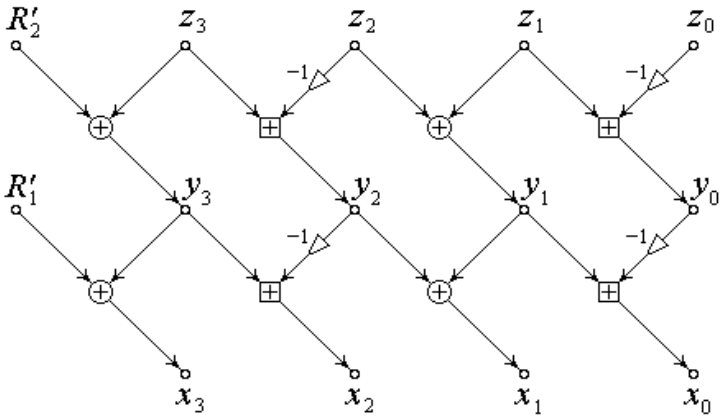


Рисунок 3.17 – Схема розшифрування ЗКК на другому раунді

Варто відзначити, що запропоновані примітиви ковзного кодування розмивають чітку структуру блоків шифруємого тексту, характерну для класичних алгоритмів симетричного блочного шифрування. З цього, зокрема, випливає, що після завершення ковзного кодування однаковим блокам відкритого тексту будуть відповідати різни (відрізняються один від одного) блоки закритого тексту. Наслидком зазначеної особливості примітивоів ковзного кодування є гіпотеза, згідно з якою може виявитись зайвим класичний режим шифрування типу *зчеплення блоків шифротексту* - ***CBC*** (*Ciphertext Block Chaining*), оскільки його функції можуть бути ріелізовані примітивом ковзного кодування.

Статистичні властивості примітивів «ковзного кодування». Будь якому цифровому файлу (тексовому, графічному, звуковому і т.і.) можна поставити у відповідність так званий *статистичний портрет*, під яким будемо розуміти гістограму розподілу числових значень байтів файлу.

Кожен байт файлу будемо розглядати, як число, і нехай n_i позначає частість байт (тобто байту, значення якого дорівнює i). Отже, цифровий

портрет файлу представляє собою послідовність чисел $n_0, n_1, ..., n_{255}$, визначающих кількість байтів певного числового значення в файлі

$$N = \sum_{i=0}^{255} n_i .$$

Позначимо як $p_i = n_i / N$ частоту появи байту з числовим значенням i (де N – обсяг файлу). Емпіричний розподіл частот випадкових чисел n_i (аналог щільності розподілу ймовірностей випадкових величин $X = n_i$) містить в собі інформацію о мірі наближеного розподілу $\{p_i\}$ до рівномірного, який можливо оцінити ентропією

$$H = -\sum_{n=0}^{255} p_i \log_2 p_i$$

Ентропія шифрограми приймає максимальне значення, рівне восьми, при $p_i = 1/256$ для всіх $i = \overline{0,255}$. Чим ближче значення H до восьми, тим вище якість розсіювання елементів алфавіту відкритого тексту. В граничному випадку, коли $p_i = 1/256$, $i = \overline{0,255}$, шифрограма набуває властивості ідеального «білого шуму».

Статистичні портрети файлів (гістограм) зручно відображати в вигляді статистичних таблиць, що містять 32 строки і вісім стовбців. Елементами таблиці є частості n_i, причому n_0 розташовується в лівому верхньому куті таблиці, а n_{255} - в правому нижньому куті. Для прикладу в табл.3.1 наведено статистичний портрет тексту, з використанням кирилиці, обсягом 208 Кбайт. Ентропія тексту, портрет якого представлено в табл.3.1, утворює величину $H = 4{,}64359$. Обсяг ущільненого (заархівованого Zip) файлу дорівнює 86 Кбайт, тобто коефіцієнт ущільнення відкритого тексту $K_1 = 208/86 = 2.42$.

Таблиця 3.1

Приклад статистичного портрету вихідного тексту

29	0	0	0	0	0	0	0
0	0	3890	0	0	3890	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
42720	35	284	0	0	0	0	2
4	4	4	0	3513	3336	2538	2
8	10	5	7	4	2	5	3
4	12	184	19	0	0	0	148
0	2	1	4	0	0	1	1
2	0	0	0	3	2	0	1
3	0	1	0	3	0	0	1
0	0	0	0	0	0	0	0
0	7	3	2	4	10	1	4
8	12	0	0	9	5	4	18
6	1	7	4	10	8	1	3
1	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
326	63	290	35	118	93	18	71
203	0	117	36	173	288	355	202
41	601	213	52	53	41	9	39
0	0	0	0	3	85	42	157
11931	2344	6368	3041	4658	12105	1484	2533
10340	1424	4787	7369	4608	9631	16730	3877
5682	8008	10169	4524	73	1411	604	2573
1354	592	28	2767	3416	403	1044	3307

Основна задача криптографічного захисту інформації, як відзначалось вище, полягає в тому, щоб перетворити осмислений вихідний текст в «білий шум», тобто зробити його рівномірно розсіяним по усьому простору елементів (букв) вхідного алфавіту з ентропією, достатньо близькою до восьми (мається на увазі 256-елементний алфавіт, що складається з усіх значень війкового байту). Запропоновані криптографічні примітиви забезпечують вельми високий рівень розсіювання, що ілюструється в табл.3.2, що містить статистичний портрет шифрограми (його вихідний портрет відображено в табл.3.1), утвореної алгоритмом арифметичного ковзного кодування. Розмір ключа складає 128 біт.

Таблиця 3.2

Статистичний портрет шифрограми

834	839	863	817	870	829	816	812
834	826	792	889	865	846	881	888
834	835	856	886	814	855	863	817
854	829	843	854	830	862	808	844
816	806	853	804	856	806	803	810
855	803	769	816	857	804	847	805
805	788	829	826	859	821	845	856
855	825	825	853	826	832	804	871
808	837	838	815	815	806	847	819
762	845	802	828	858	865	831	877
899	811	885	872	834	804	835	832
782	844	828	838	809	830	841	849
830	799	818	818	870	850	839	848
804	859	852	833	898	819	864	838
887	803	860	905	849	814	825	753
810	853	833	806	853	832	859	874
855	850	876	829	832	843	833	833
838	871	825	842	829	789	886	814
802	811	859	890	816	810	805	870
805	820	810	851	851	836	865	851
852	805	858	861	844	819	824	798
842	825	787	801	806	780	858	855
829	822	851	886	775	836	822	822
803	821	826	849	854	833	831	890
802	837	821	756	816	813	834	836
796	851	859	794	844	818	806	878
829	820	810	814	801	803	770	875
839	869	849	785	858	820	837	863
805	901	860	829	860	860	850	808
804	882	865	838	860	843	794	873
934	825	849	821	826	850	773	848
823	825	855	813	832	881	877	854

Ентропія шифрограми $H = 7{,}99915$. Близькими до якості розсіювання виявились також алгоритми логічного і змішаного ковзного кодування.

Шифрограми, що формуються методами ковзного кодування, практично не ущільнюються, що свідчить про високу якість криптоперетворення, що забезпечується даним примітивом.

Тривимірне «ковзне кодування». В попередніх підрозділах було розглянуто лівостороннє, правостороннє і змішане ковзне кодування. Це функціональні особливості. Але існують також і структурні особливості ковзного кодування, що полягають у представленні інформаційної послідовності у вигляді певної структурі, наприклад куба або тривимірної матриці. За рахунок збільшення мірності даних з'являються нові властивості в кодуванні. Такі як напрямки кодування. На рис.3.18 зображено, що в тривимірному просторі поняття лівостороннє, правостороннє і змішане ковзне кодування ще має уточнення в вигляді просторового напрямку за яким воно відбувається. Осі x, y, z для тривимірних матриць зазвичай позначають як напрямки i, j, k.

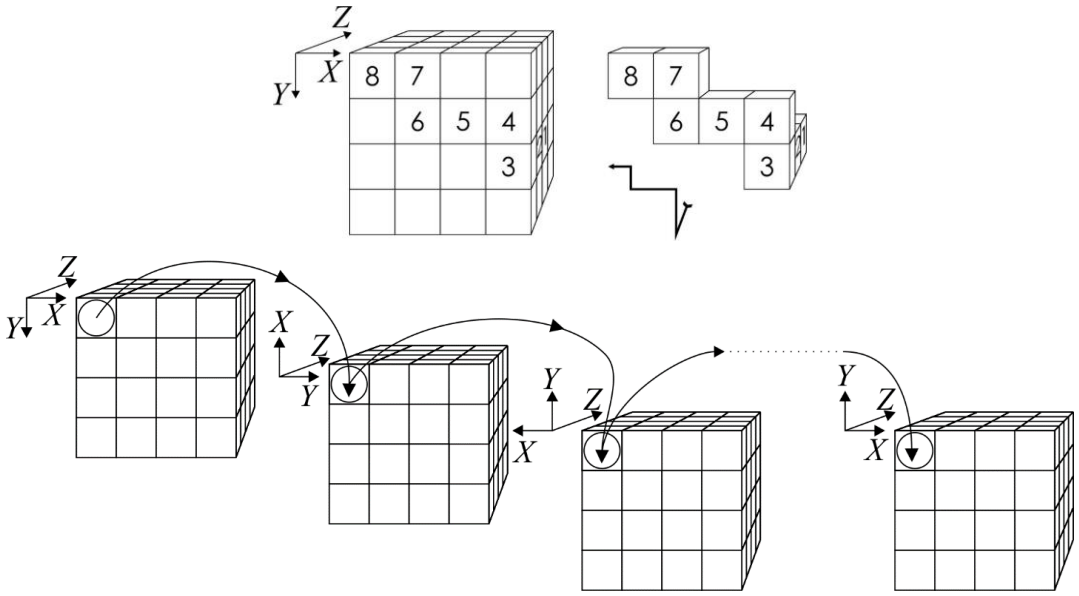


Рисунок 3.18 – Напрямки ковзного кодування при структуруванні даних у вигляді тривимірної матриці

Обрання певного напрямку не потребує витрат машинного часу і складних обрахунків. Але використання додаткових параметрів для шифрування збільшує якість відбілювання шифрограми.

3.3. Криптографічний примітив «3D matrix multiplications»

Найпростіший варіант множення матриць 3D це обрання певного напрямку і множення по перетинах (рис.3.19).

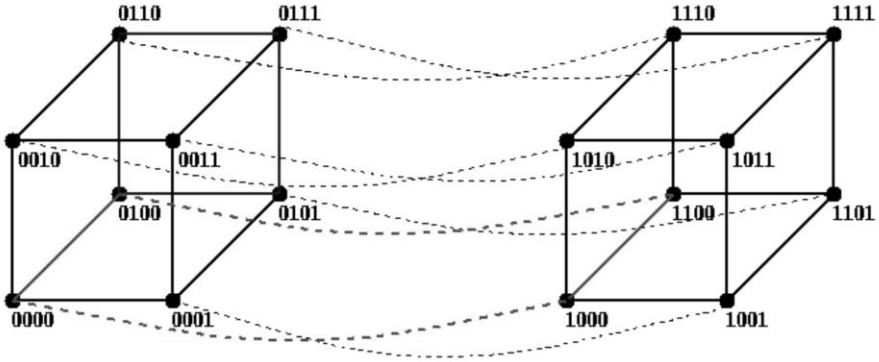


Рисунок 3.19 – Множення матриць по перетинам за обраним напрямком

Таким чином можна записувати ключ як масив звичайних матриць. Цей масив і буде матрицею 3D.

Тривимірне множення матриць. Використовуючи математичнй апарат наведений в другому розділі синтезовано криптографічний примітив, який виконує просторове множення матриць. При чому ключ впливає на генерування невироджених тривимірних матриць, які є матрицями ключа. В подальшому відбувається множення матриці даних на матрицю ключа, обидві матрицю мають тривимірну структуру. При розшифрування знаходяться обернені матриці для ключа розшифрування.

3.4. Криптографічний примітив «Substitutions»

Зазвичай в криптографічних примітивах нелінійної заміни з використанням таблиці підстановок SBox виконується принцип, що данні які

підлягають заміні представляються у вигляді двох рівних частин які визначають координати в SBox.

$$P = p_{row} \vee p_{col},. \quad (3.4)$$

де P – якісь данні (байт даних) які потрібно замінити; p_{row} і p_{col} – дві складові даних (наприклад, старший і молодший півбайти).

Наприклад, якщо потрібно замінити байт $P = 1F$ (шістнадцятковий вигляд), то для нього обираються координати рядку $p_{row} = 1$ і стовбця $p_{col} = F$. За цими координатами в таблиці підстановок знаходиться значення C, яке заміняє значення P.

Для пидвіщення криптографічної стійкості автор пропонує використовувати випадкові координати які утворюються тим, що номер строки і стовбця береться довільно в максимально можливому діапазоні значень. Для байту 4 біта відповідають за номер строки і 4 біта за номер стовбця. Виходячи з цього пропонується метод побудови S-блоків більших розмірів. Якщо «класичні» таблиці мають квадратну форму розміром 4x4 або 16x16, то тепер потрібна таблиця 32x32, яка містить 1024 значення. Перехід від 256 до 1024 значень може здатись кращим, але розрахунки над НП 8 і 10 ступеню потребують різних часових витрат для формування SBox. Тому таблиці більшого розміру можна синтезувати з декількох таблиць меншого розміру не вдаючись до складних розрахунків.

Як видно на рис.3.20. можна використовувати одночасно чотири різні SBox і при цьому адресація зміниться з чотирьохрозрядної на п’ятирозрядну. Наприклад треба замінити $P = 1F$. Як було показано вище, то при використанні SBox розміром 16x16 треба взяти по чотири розряди для визначення рядка и стовбця (координат в таблиці замін), тобто $1F = 00011111$ де $1 = 0001$ і $F = 1111$. Якщо ж використати SBox 32x32, то можна взяти за координати 00011 і 11111, що відповідно $row = 3$ і $col = 1F$. Як було зазначено вище, обирати координати можна в довільному порядку.

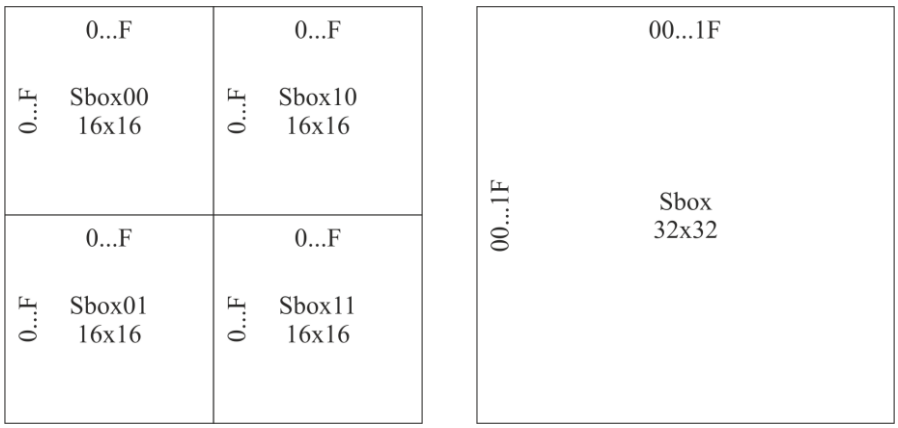


Рисунок 3.20 – SBox розміром 32x32, що складається з чотирьох SBox розміром 16x16

Поширюючи цей принцип далі отримуємо SBox 256x256, для побудови якого використано 256 SBox 16x16.

Якщо розглядати $P = 1F = 00011111$, то стає видно що $row = col = 1F$, але можна брати довільну кількість розрядів для визначення номера рядка і строки в SBox. Тобто значення може знаходитись в діапазоні $0 \ldots 1F$, в залежності від того скільки розрядів ми обираємо для визначення номера рядка чи строки.

Запропонований метод можна модифікувати обираючи не тільки довільну кількість розрядів для визначення номера строки і стовбця в SBox, а ще й вибираючи їх в «довільній» послідовності. Наприклад, парні розряди відповідають за номер рядка, а непарні за номер стовбця. Так само можна обирати розряди в зворотній послідовності роздивляючись $1F$ як $F1$ чи щось інше (це визначається конкретною реалізацією цього методу).

Метод побудови SBox, що базується на перехід від 2D до 3D полягає в тому, що обрання потрібних даних відбувається за більшою системою координат (рис.3.21 і рис.3.22). Можна обирати скільки завгодно мірні системи координат, але перехід від 2D до 3D більш наглядно демонструє переваги багатомірності.

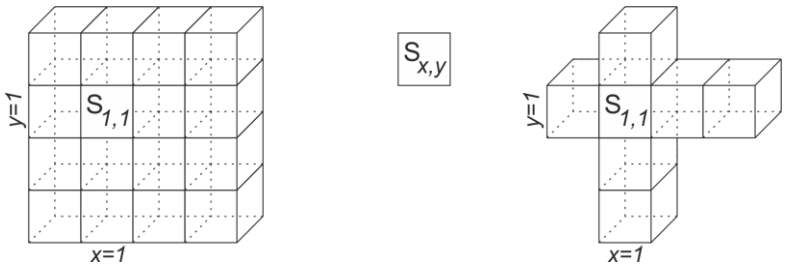


Рисунок 3.21 – Схема обирання в SBox 2D за певними координатами

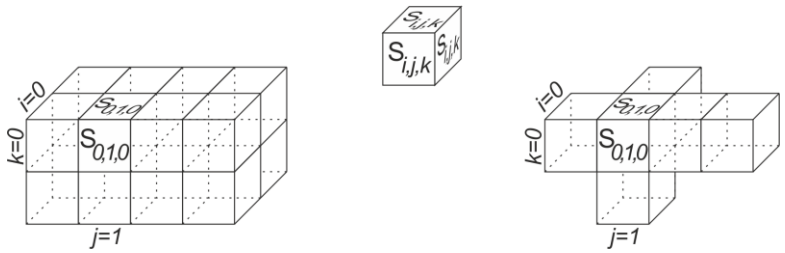


Рисунок 3.22 – Схема обирання в SBox 3D за певними координатами

Як видно з рис.3.21 і рис.3.22. при однаковому розмірі SBox ($4 \times 4 = 2 \times 2 \times 4$) що містить 16 значень (для полегшення зображення і пояснень) потрібна різна кількість координат. Відповідна більша кількість груп розрядів що визначають положення даних для заміни.

Перемішування таблиць підстановок. Структура типової таблиці замін SBox для AES розміром 16x16 має вигляд 2D квадрату.

Цю саму таблицю можна записати у 3D вигляді декількома варіантами, наприклад як таблицю розміром 4x4x16 або таблицю розміром 4x8x8 рис. 3.23:

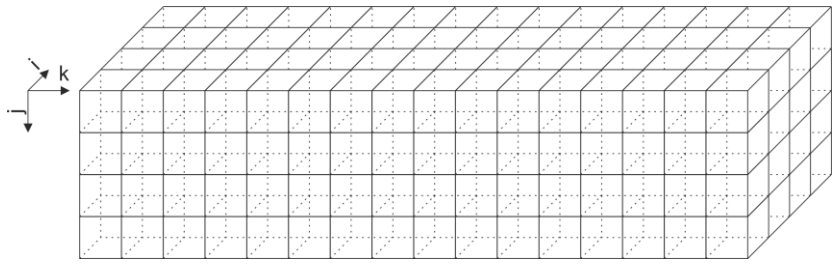


Рисунок 3.23 – SBox розміром 4x4x16

Якщо розділити SBox 4x4x16 на блоки, то він матиме такий вигляд рис.3.24:

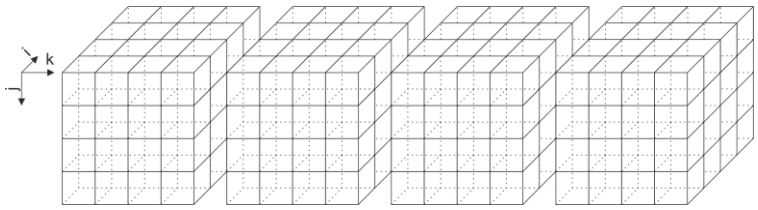


Рисунок 3.24 – SBox розміром 4x4x16 розділений на чотири блоки розміром 4x4x4

Умовно можна позначити, що відбувся перехід як на рис.3.24, за винятком того, що за одиницю даних береться більш складна структура, а саме тривимірна матриця розміром 4x4x4.

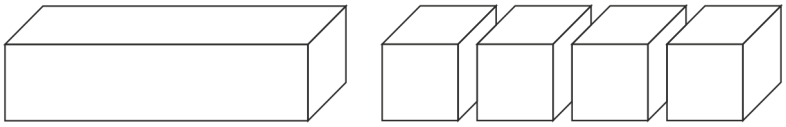


Рисунок 3.25 – Розділення даних на блоки

Отже, маємо перехід від 1D до 3D де кожна одиниця даних являє собою вже набір даних структурованих в 3D. Тобто можна в 3D структурувати данні по різному використовуючи мікро- і макроструктури рис.3.26.

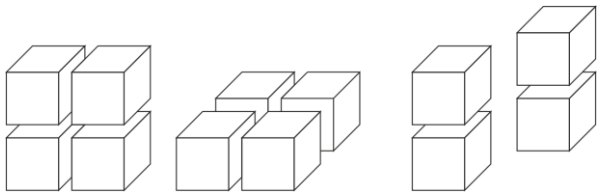


Рисунок 3.26 – Декілька можливих комбінацій з чотирьох кубиків даних

Як було зазначено вище, існує багато варіантів представлення таблиці SBox в тривимірному вигляді. Розглядаємо ту саму таблицю розміром 256x256, але в 3D:

Умовно їх можна позначити як рис.3.27 і рис.3.28:

Рисунок 3.27 – Еквівалентне зображення SBox 3D що містить 256 значень

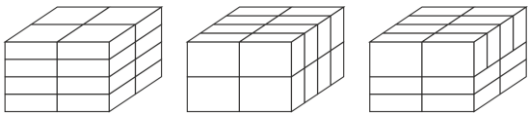


Рисунок 3.28 – Формування «кубиків» (макроблоків) з SBox 3D

Збираючи таким чином «кубік» можна отримати SBox розміром 256x256x256. Якщо рахувати за одиницю даних один байт то вийде 16777216 байт, або приблизно 16Мб. Для сучасних ПК такий розмір є незначним і можна використовувати зразу весь SBox, але для МК це досить суттєвий об’єм даних. Перевага зазначеного підходу як раз і полягає у тому, що можна формувати будь яку частину макро SBox виконуючи різні перетворення над SBox меншого розміру. Тобто формувати макро SBox з модифікованих SBox меншого розміру. При цьому витрачаючи тільки 256 байт на зберігання таблиці.

Модифікація SBox. При зашифрвуванні і розшифрвуванні використовуються SBox і обернений SBox, які містять випадкову послідовність чисел, що не повторюються. Фактично, для того щоб SBox добре працював в шифраторі треба щоб кожного разу один і той самий байт даних замінявся на нове значення. Легше всього це робити зсуваючи усю таблицю замін на одну позицію (чи на якусь непарну кількість позицій). Це обумовлюється ключем. Таким чином отримуємо з одного SBox послідовність в 255 «різних» SBox. В будь якому разі потрібно сформувати початковий SBox який залежний від ключа шифрування. Ключ шифрування зазвичай 256 біт або 32 байти, для SBox потрібно 256 байт, тобто потрібна мінімум в 8 раз довша псевдовипадкова послідовність ніж є в ключі. Потрібно розширити ключ до розміру 256 байт.

Формування SBox з ключа. Маємо випадковий ключ довжиною 256 біт з нього потрібно отримати 2048 біт що складають 256 байт. При цьому байти не повинні повторюватись у послідовності. Схему формування SBox розміром 256 байт (16x16) з ключа розміром 32 байта наведено на рис рис.3.29.

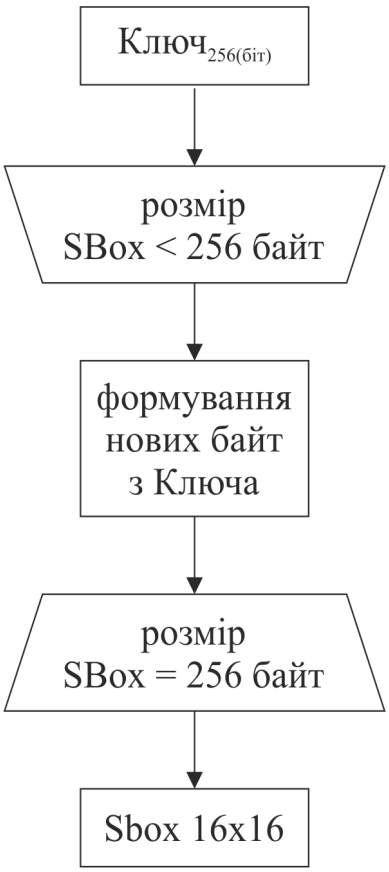


Рисунок 3.29 – Схема формування SBox розміром 256 байт

Для пояснення суті методу зазначеного на рис.3.29 використовуємо SBox розміром 4x4, що формується з півбайт (рис.3.30). Ключ при цьому має довжиною 16 біт.

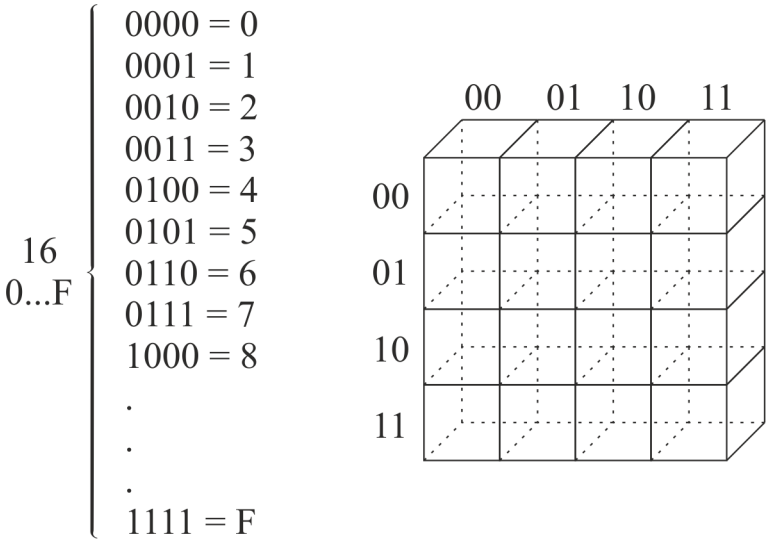


Рисунок 3.30 – Формування SBox з півбайт

Наприклад, ключ формується з послідовності 0,1,2,3. З нього треба отримати послідовність від 0 до 15 рис.3.31, рис.3.32 і рис3.33.

i = 0: **0000 0001 0010 0011** = Ключ(старт)$_4$
i = 1: ~~0000 0010~~ 0100 0110
i = 2: ~~0000 0100~~ 1000 1100
i = 3: ~~0000~~ 1001 ~~0001 1000~~
i = 0: **0100 0110 1000 1100 1001** = Ключ(1)$_5$
i = 1: ~~1000~~ 1101 ~~0001 1001 0010~~
i = 2: ~~0001~~ 1010 ~~0011 0010~~ 0101
i = 3: ~~0011 0100 0110 0100 1010~~
i = 0: **1101 1010 0101** = Ключ(2)$_3$
i = 1: 1011 ~~0100 1011~~
i = 2: ~~0110 1001 0110~~
i = 3: ~~1101 0010 1101~~
i = 0: **1011** = Ключ(3)$_1$
i = 1: 0111
i = 2: 1110
i = 3: ~~1101~~
i = 0: **0111 1110** = Ключ(4)$_2$
i = 1: 1111 ~~1100~~
i = 2: ~~1111 1001~~
i = 3: ~~1111 0011~~
i = 0: **1111** = Ключ(5)$_1$

Рисунок 3.31 – Побітовий зсув вліво для формування ключа довжиною 16 півбайт

$$Ключ_{16} = Ключ(старт)_4 \vee Ключ(1)_5 \vee Ключ(2)_3 \vee Ключ(3)_1 \vee Ключ(4)_2 \vee Ключ(5)_1 \quad (3.5)$$

Цей приклад показує, що можна отримати групу чисел. Розширюючи ключ малої довжини до ключа більшої довжини. При цьому є обмеження. З ключа довжиною 16 біт можна отримати 16х4=64 біт, з ключа 256 біт можна отримати 256х8=2048 біт. Мається на увазі, що при формуванні SBox що складається з півбайт (4 біта) можемо ключ збільшити в 4 рази. При формуванні SBox що складається з байтів (8 біт) можемо ключ збільшити в 8 разів.

Таким чином ми отримали ключ:

0000	Ключ(старт)$_4$	0 (00; 00)
0001		1 (00; 01)
0010		2 (00; 10)
0011		3 (00; 11)
0100	Ключ(1)$_5$	4 (01; 00)
0110		6 (01; 10)
1000		8 (10; 00)
1100		C (11; 00)
1001		9 (10; 01)
1101	Ключ(2)$_3$	D (11; 01)
1010		A (10; 10)
0101		5 (01; 01)
1011	Ключ(3)$_1$	B (10; 11)
0111	Ключ(4)$_2$	7 (01; 11)
1110		E (11; 10)
1111	Ключ(5)$_1$	F (11; 11)

Рисунок 3.32 – Формування ключа більшої довжини

Отримано ключ $Ключ_{16} = 0123468C9DA5B7EF$, початок ключа співпадає зі стартовим ключем $Ключ(старт)_4 = 0123$.

Sbox

	00	01	10	11
00	**0**	**1**	**2**	**3**
01	**4**	6	8	C
10	9	D	A	5
11	B	7	**E**	**F**

$\overline{\text{Sbox}}$

	00	01	10	11
00	**0**	**1**	**2**	**3**
01	**4**	B	5	D
10	6	8	A	C
11	7	9	**E**	**F**

	00	01	10	11
00	**00 00**	**00 01**	**00 10**	**00 11**
01	**01 00**	01 10	10 00	11 00
10	10 01	11 01	10 10	01 01
11	10 11	01 11	**11 10**	**11 11**

	00	01	10	11
00	**00 00**	**00 01**	**00 10**	**00 11**
01	**01 00**	10 11	01 01	11 01
10	01 10	10 00	10 10	11 00
11	01 11	10 01	**11 10**	**11 11**

Рисунок 3.33 – Формування SBox з розширеного ключа

Формування SBox з незвідного поліному (НП) в полі GF(2). Для прикладу розглянемо використання НП четвертного ступеню. В подальшому всі ці дії поширяться на НП восьмого ступеня і шістнадцятирічну систему

числення. Деякі уточнення, НП четвертого ступеня $f_4 = 10011$ це те саме що $f_4 = x^4 + x + 1$ тільки записаний в степеневому вигляді, утворюючий елемент $\varpi = 10$ це $\varpi = x$. Оскільки розрахунки ведуться в GF(2), то 1 і 0 це можливі коефіцієнти перед x^n в многочлені $f_4 = 1 \cdot x^4 + 0 \cdot x^3 + 0 \cdot x^2 + 1 \cdot x + 1$. Створюється мультиплікативна група $\varpi^k (\mathrm{mod}\,\varphi)$, де $k = k_2 \perp k_1$, а $\varphi = f_4$.

Добуток поліному в кільці (один зі способів множення):

$f_4 = 10011$ $\varpi = 10$

$10^k (\mathrm{mod}10011)$

1. $\varpi^0 = (00010)^0 = (00001)$;
2. $\varpi^1 = \varpi^0 \cdot \varpi = (00010)^1 = (00001) \cdot (00010) = (00010)$;
3. $\varpi^2 = \varpi^1 \cdot \varpi = (00010)^2 = (00010) \cdot (00010) = (00100)$;
4. $\varpi^3 = \varpi^2 \cdot \varpi = (00010)^3 = (00100) \cdot (00010) = (01000)$;
5. $\varpi^4 = \varpi^3 \cdot \varpi = (00010)^4 = (01000) \cdot (00010) = (10000)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^4 = \oplus \begin{matrix} 10000 \\ 10011 \end{matrix} = 00011$;
6. $\varpi^5 = \varpi^4 \cdot \varpi = (00010)^5 = (00011) \cdot (00010) = (00110)$;
7. $\varpi^6 = \varpi^5 \cdot \varpi = (00010)^6 = (00110) \cdot (00010) = (01100)$;
8. $\varpi^7 = \varpi^6 \cdot \varpi = (00010)^7 = (01100) \cdot (00010) = (11000)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^7 = \oplus \begin{matrix} 11000 \\ 10011 \end{matrix} = 01011$;
9. $\varpi^8 = \varpi^7 \cdot \varpi = (00010)^8 = (01011) \cdot (00010) = (10110)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^8 = \oplus \begin{matrix} 10110 \\ 10011 \end{matrix} = 00101$;
10. $\varpi^9 = \varpi^8 \cdot \varpi = (00010)^9 = (00101) \cdot (00010) = (01010)$;
11. $\varpi^{10} = \varpi^9 \cdot \varpi = (00010)^{10} = (01010) \cdot (00010) = (10100)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^{10} = \oplus \begin{matrix} 10100 \\ 10011 \end{matrix} = 00111$;
12. $\varpi^{11} = \varpi^{10} \cdot \varpi = (00010)^{11} = (00111) \cdot (00010) = (01110)$;

13. $\varpi^{12} = \varpi^{11} \cdot \varpi = (00010)^{12} = (01110) \cdot (00010) = (11100)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^{12} = \oplus \begin{matrix} 11100 \\ 10011 \end{matrix} = 01111$;

14. $\varpi^{13} = \varpi^{12} \cdot \varpi = (00010)^{13} = (01111) \cdot (00010) = (11110)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^{13} = \oplus \begin{matrix} 11110 \\ 10011 \end{matrix} = 01101$;

15. $\varpi^{14} = \varpi^{13} \cdot \varpi = (00010)^{14} = (01101) \cdot (00010) = (11010)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^{14} = \oplus \begin{matrix} 11010 \\ 10011 \end{matrix} = 01001$;

16. $\varpi^{15} = \varpi^{14} \cdot \varpi = (00010)^{15} = (01001) \cdot (00010) = (10010)$ це значення вже тої ж розрядності (ступеню) що і $f_4 = 10011$ поэтому $\varpi^{15} = \oplus \begin{matrix} 10010 \\ 10011 \end{matrix} = 00001$; Але!!!

На цій ітерації отримано початкове $\varpi^0 = \varpi^{15} = 00001$, подальше возведення у ступінь призведе до отримання того ж самого значення, наприклад, $\varpi^1 = \varpi^{16}$ або $\varpi^2 = \varpi^{17}$, $\varpi^3 = \varpi^{18}$, $\varpi^4 = \varpi^{19}$, … $\varpi^0 = \varpi^{15} = \varpi^{30} = 00001$ і т.д.

В результаті отримано мультиплікативну групу довжиною 16 значень. Ось ці значення в тій послідовності, як вони були знайдені:

00001; 00010; 00100; 01000;

00011; 00110; 01100; 01011;

01010; 00101; 00111; 01110;

01111; 01101; 01001; 00001.

Запишемо їх у табл. 3.3 розміром 4х4. Зліва направо і зверху донизу.

Таблиця 3.3

k_2 \ k_1	00	01	10	11
00	00 01	00 10	01 00	01 00
01	00 11	01 10	11 00	10 11
10	10 10	01 01	01 11	11 10
11	11 11	11 01	10 01	00 01

Оскільки вийшла таблиця 4х4, то вона зручніше записується у четвірковій системі числення.

00 = *0* (00000); 01 = *1* (00001); 02 = *2* (00010); 03 = *3* (00011);

10 = *4* (00100); 11 = *5* (00101); 12 = *6* (00110); 13 = *7* (00111);

20 = *8* (01000); 21 = *9* (01001); 22 = *10* (01010); 23 = *11* (01011);

30 = *12* (01100); 31 = *13* (01101); 32 = *14* (01110); 33 = *15* (01111);

100 = *16* (10000);

Зліва від знаку « = » четвіркове значення, праворуч від знаку « = » десяткове значення, а в дужках двійкове значення).

У четвірковій системі отримано послідовність:

01; 02; 10; 20; 03; 12; 30; 23; 11; 22; 13; 32; 33; 31; 21; 01

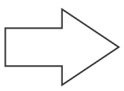
Інверсія (запис у зворотному порядку) має вигляд:

01; 21; 31; 33; 32; 13; 22; 11; 23; 30; 12; 03; 20; 10; 02; 01

Запишемо отриману послідовність зліва у табл. 3.4 і справа її інверсію.

Таблиця 3.4

k_2 \ k_1	0	1	2	3
0	01	02	10	20
1	03	12	30	23
2	11	22	13	32
3	33	31	21	01



k_2 \ k_1	0	1	2	3
0	01	21	31	33
1	32	13	22	11
2	23	30	12	03
3	20	10	02	01

Оскільки перше і останнє значення повторюються, то потрібно зробити зсув всіх значень в таблиці на одну позицію вправо. Після цього саме останнє значення 01 зникає, а саме перше значення в таблиці стає 00.

Перезаписавши послідовність (і інвертовану послідовність) добавляється 00 на початок і убирається 01 в кінці, як написано віще. Після цих дій отримано два масиви:

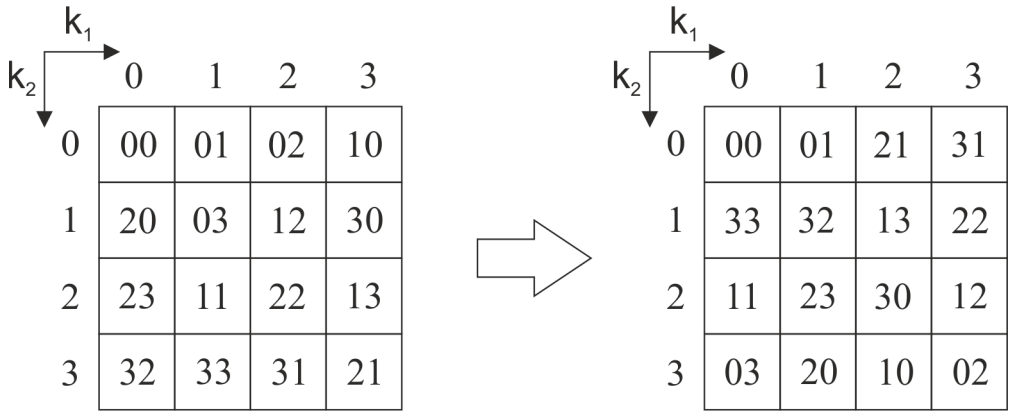
Масив1 = {00; 01; 02; 10; 20; 03; 12; 30; 23; 11; 22; 13; 32; 33; 31; 21}

i

Масив2 = {00; 01; 21; 31; 33; 32; 13; 22; 11; 23; 30; 12; 03; 20; 10; 02}

Запишемо їх у табл. 3.5. Ліворуч Масив1 і праворуч Масив2.

Таблиця 3.5



Наступна операція «ранжування».

Масив_МО = значення Масиву1 за індексом Масиву2.

```
for (int i = 0; i < довжина масиву; i++)
{
        Масив_МО[ Масив1[i] ] = Масив2[i];
}
```

В цьому циклі формується мультиплікативна зворотна група, що зберігається в Масив_МО.

Масив_МО = {00; 01; 21; 32; 31; 23; 13; 12; 33; 02; 30; 11; 22;10; 03; 20}

Тепер може сформуватись таблиця мультиплікативної зворотньої величини x^{-1}. Запис значень з Масив_МО в табл.3.6 розміром 4х4

Таблиця 3.6

k_2 \ k_1	0	1	2	3
0	00	01	21	32
1	31	23	13	12
2	33	02	30	**11**
3	22	10	03	20

Розраховується таблиця мультиплікативних зворотних (МО) величин $(x \oplus \alpha)^{-1}$ $\alpha = 1011_2 = 23_4$. Для цього в клітинку $x = 00$ перетягується значення 11, що відповідає числу $x = 23$. Всі інші елементи визначаються циклічним зсувом попередньо наведених елементів, визначаючи 11 лідером ланцюга. Іншими словами, зсув «змійкою» усіх значень таблиці вліво так, щоб початкове значення стало 11. Усі числа з таблиці, які йшли до 11 переносяться підряд в кінець таблиці. Запишемо нові результати у табл.3.7.

Таблиця 3.7

k_2 \ k_1	0	1	2	3
0	**11**	22	10	03
1	20	00	01	21
2	32	31	23	13
3	12	33	02	30

Далі синтезується матриця A для $f_4 = 10011$ і $\theta = 101$ діагональним заповнення і приведенням до залишку за модулем $f_4 = 10011$.

$$A=\begin{pmatrix}1 & 1 & 1 & 0\\ 0 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\end{pmatrix}$$

Важливо, щоб матриця А була невироджена, інакше буде неможливо отримати зворотній SBox, який потрібен для розшифрування.

Розраховується таблиця $y=(x\oplus\alpha)^{-1}\otimes A$ матричним множенням попередніх елементів на A, записуючи чотири розряди, що формуються в результаті множення, в четвірковій формі.

Наприклад, для клітинки з координатами 00. Множення 11 на матрицю А. 11 в четвірковому вигляді – 0101 в двійковому вигляді.

$$\begin{vmatrix}0 & 1 & 0 & 1\end{vmatrix}\times\begin{vmatrix}1 & 1 & 1 & 0\\ 0 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\end{vmatrix}=\begin{vmatrix}0 & 2 & 1 & 2\end{vmatrix}(\mathrm{mod}\,2)=\begin{vmatrix}0 & 0 & 1 & 0\end{vmatrix}=02$$

Наприклад, для клітинки з координатами 01. Множення 22 на матрицю А. 22 в четвірковому вигляді – 1010 в двійковому вигляді.

$$\begin{vmatrix}1 & 0 & 1 & 0\end{vmatrix}\times\begin{vmatrix}1 & 1 & 1 & 0\\ 0 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\end{vmatrix}=\begin{vmatrix}2 & 1 & 2 & 0\end{vmatrix}(\mathrm{mod}\,2)=\begin{vmatrix}0 & 1 & 0 & 0\end{vmatrix}=10_4$$

Наприклад, для клітинки з координатами 02. Множення 10 на матрицю А. 10 в четвірковому вигляді – 0100 в двійковому вигляді.

$$\begin{vmatrix}0 & 1 & 0 & 0\end{vmatrix}\times\begin{vmatrix}1 & 1 & 1 & 0\\ 0 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\end{vmatrix}=\begin{vmatrix}0 & 1 & 1 & 1\end{vmatrix}=13_4$$

В результаті таких дій, формується набір елементів, запишемо їх у табл 3.8.

Таблиця 3.8

k_2 \ k_1	0	1	2	3
0	**02**	**10**	**13**	33
1	32	00	11	23
2	03	30	01	20
3	31	12	22	21

Формується елементи SBox як результат послідовності обчислень

$y = (x \oplus \alpha)^{-1} \otimes A \oplus \beta$ $\beta = 0110_2 = 12_4$ і записуємо їх у табл.3.9.

Таблиця 3.9

k_2 \ k_1	0	1	2	3
0	10	02	01	21
1	20	12	03	31
2	11	22	13	32
3	23	00	30	33

На останньому кроці створюється зворотній SBox шляхом підстановки виду: якщо в SBox по координатам рядок 1 і стовбець 2 знаходиться значення 03, то в таблиці зворотний SBox в рядку 0 і стовбці 3 знаходиться значення 12 і навпаки.

Нелінійні властивості підстановки що прослідковуються у SBox, показано у вигляді графіку на рис.3.34.

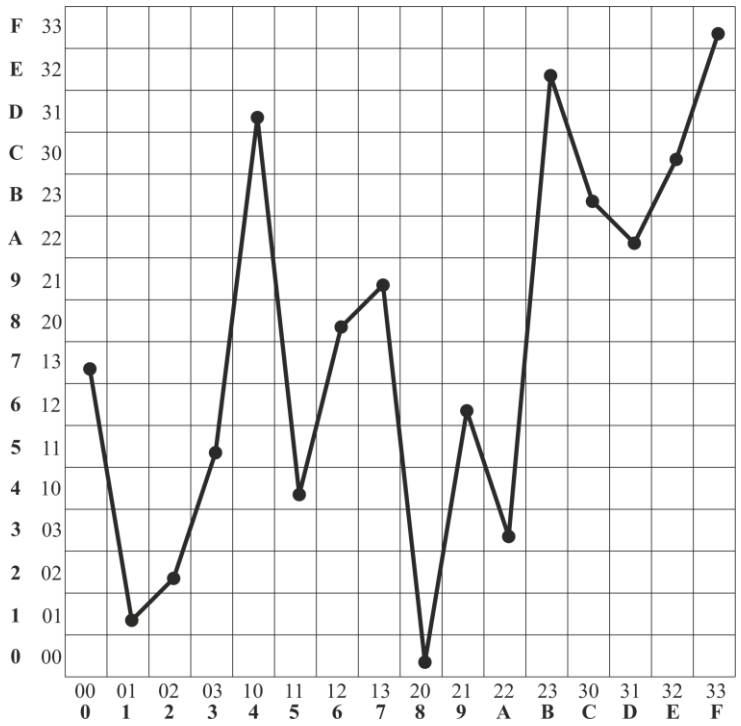


Рисунок 3.34 – Статистична залежність SBox від оберненого SBox

Синтез SBox, який містить в собі інші SBox. Вище були описані варіанти формування SBox, наступним кроком для формування криптографічного примітиву «Substitutions» є формування масиву з різних (унікальних SBox). Після чого виконуються перестановки всередині блоку і формуються нові блоки SBox 3D (рис.3.35 і рис.3.36)

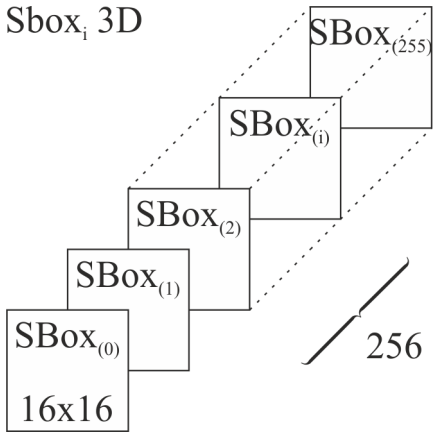


Рисунок 3.35 – Блок SBox 3D, що містить 256 різних SBox розміром 16x16

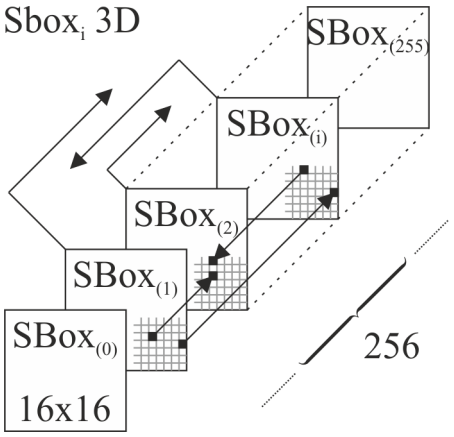


Рисунок 3.36 – Перестановки всередині блоку SBox 3D

3.5. Рівномірно щільні примітиви нелінійної підстановки

Синтез рівномірно щільних S-блоків. Класичним примітивом нелінійної підстановки (ПНП) може бути названо S-блок симетричного блочного шифру AES, який здійснює афінне перетворення

$$y = x_f^{-1} \cdot A + b \tag{3.7}$$

Діаграма розсіювання S-блока AES шифру представлена на рис.3.37.

Візуальний огляд показує що, діаграми розсіювання S-блоку шифру AES далека від діаграми з рівномірною щільністю. При тому, що середня кількість точок $\bar{n}$ в елементах поділу діаграми розсіювання (порядок яких складає 32x32) повинно дорівнювати чотирьом, фактично в елементи потрапляють від нуля до десяти точок. Середньо квадратичне відхилення σ кількості точок, що містяться в елементах діаграми розсіювання, дорівнює 5.12, а коефіцієнт кореляції $r = -0.0438$.

Алгебраїчний метод опису S-блоку шифру AES надає примітиву слабкості, суть якої полягає в наступному. Афінне перетворення (3.7) обумовлює можливість апроксимації ПНП системою булевих рівнянь, що, в свою чергу, ставить примітив під загрозу реалізації алгебраїчної атаки [3]. З метою знешкодження уразливості до алгебраїчної атаки в останній час

переходять від детермінованих алгоритмів синтезу примітивів на основі афінних перетворень до стохастичних методів побудови S-блоків.

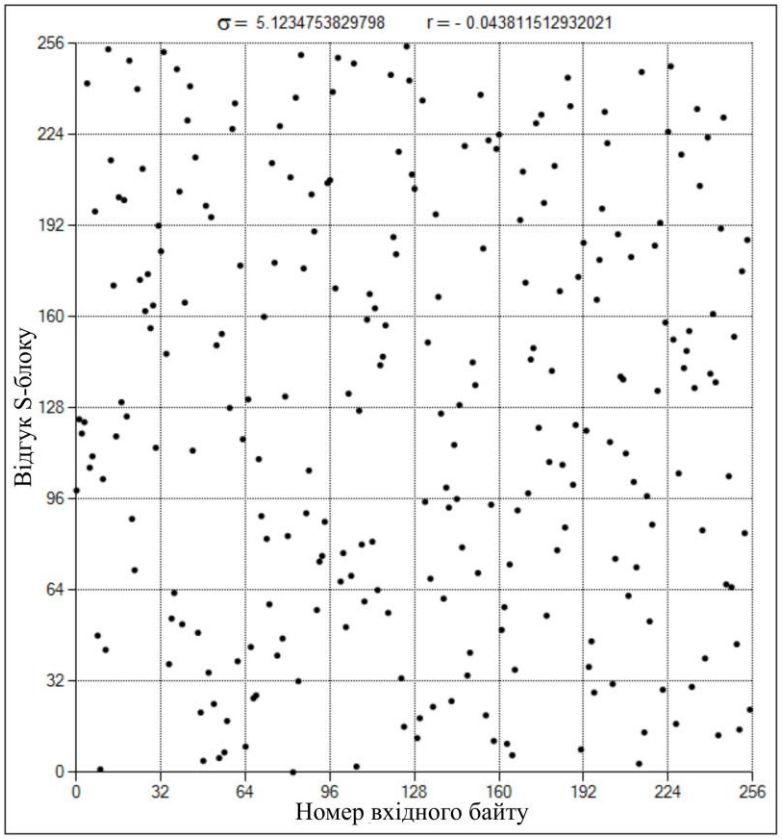


Рисунок 3.37 – Діаграма розсіювання примітиву нелінійної підстановки алгоритму AES

Такий спосіб використовується, в тому числі, при розробці S-блоків Українського національного стандарту симетричного блочного шифрування ДСТУ 7624:214 та в інших шифрах.

Пропонується такий метод синтезу рівномірно щільних ПНП «байт-в-байт», який будемо називати методом синтезу S-блоків з вибуванням. Суть методу полягає у такому.

Нехай X і Y – вхідні і вихідні байти стохастичного S-блоку відповідно, десяткові еквіваленти яких лежать в діапазоні від 0 до 255. Розділимо шкали діаграм розсіювання X і Y, названі на рис.3.37 як «Номер вхідного байту» і «Відгук S-блоку», на вісім рівномірних інтервалів, кожен з яких містить по 32 цілочислених відліку. Стохастичне заповнення квадратів будем виконувати

таким чином, щоб кожен квадрат розділу $S_{k,l}$, $k,l=\overline{1,8}$, де k – номер строки, а l – номер стовбця діаграми розсіювання, містив по чотири точки з координатами $(X,Y)\in S_{k,l}$ при цьому квадрат $S_{1,1}$ розташуємо біля початку координат діаграми.

Домовимось вважати, що якщо точка (X,Y) розташована на лівій боковій грані квадрата $S_{k,l}$ або на його основі, то вона належить цьому квадрату. Обираємо, як найбільш простий, порядок заповнення діаграми розсіювання по стовбцям. Утворюємо два допоміжних вектори довжиною 256 байт, а саме, вектор X, в комірку якого внесені восьмибітні числа від 0 до 255 і вектор Y, всі комірки якого обнулені.

Спочатку генерується випадкове рівномірне розподілення в інтервалі від 0 до 1 число (РРЧ) x і вираховується ордината $y_0=[x\cdot 256]$, де $[\alpha]$ – ціла частина значення α.

Байт числа y_0 розташовується в нульовій комірці вектору Y, тобто в комірці $Y(0)$, а комірка $X(y_0)$ виключається з вектору X. Потім генерується наступне РРЧ x і розраховується ордината $y_1=[x\cdot 255]$. Число, що міститься в комірці $X(y_1)$, розташовується в комірці $Y(1)$, а комірки $X(y_1)$ виключаються з вектору X і т.д. Отже, на і-му кроці генерації РРЧ x і-ю комірку вектору Y записується байт числа $y_i=[x\cdot N_i]$, де $N_i=256-i$, також виключається комірка $X(y_i)$ вектору X.

Якщо на деякому і-му кроці генерації РРЧ x з’ясується, що в якому небуть квадраті першого стовбця діаграми розсіювання знаходиться чотири точки, то всі інші 28 комірок вектору X цього квадрату тимчасово виключається з розгляду.

Перед заповненням квадратів другого стовбця таблиці розсіювання поновлюються 224 комірки вектору X, тобто початковий вектор X за виключенням тих 32 комірок, які були задіяні на етапі формування першого

стовбця таблиці. Так само перехід до заповнення кожного наступного стовбця діаграми розсіювання передбачає виключення з вектору X 32 комірок, використаних при формуванні попереднього стовбця таблиці. Початкове значення індексу i для квадратів другого стовбця дорівнює 32, для третього – 64 і т.д.

Таким чином, після заповнення усіх стовбців діаграми розсіювання вектору Y буде містити усю інформацію відносно стохастично модульованого примітиву нелінійної підстановки. При цьому номер комірки $x = \overline{0,255}$ вектору Y є аргументом, а вміст комірки $Y(x) = y \in [0,255]$ – функцією f нелінійного перетворення $y = f(x)$.

Приклад результатів стохастичного моделювання рівномірно щільного S-блоку показано на рис.3.38.

Особливість такого блоку полягає в тому, що на діаграмі розсіювання відсутні пусті клітини і всі клітини мають рівну щільність байт.

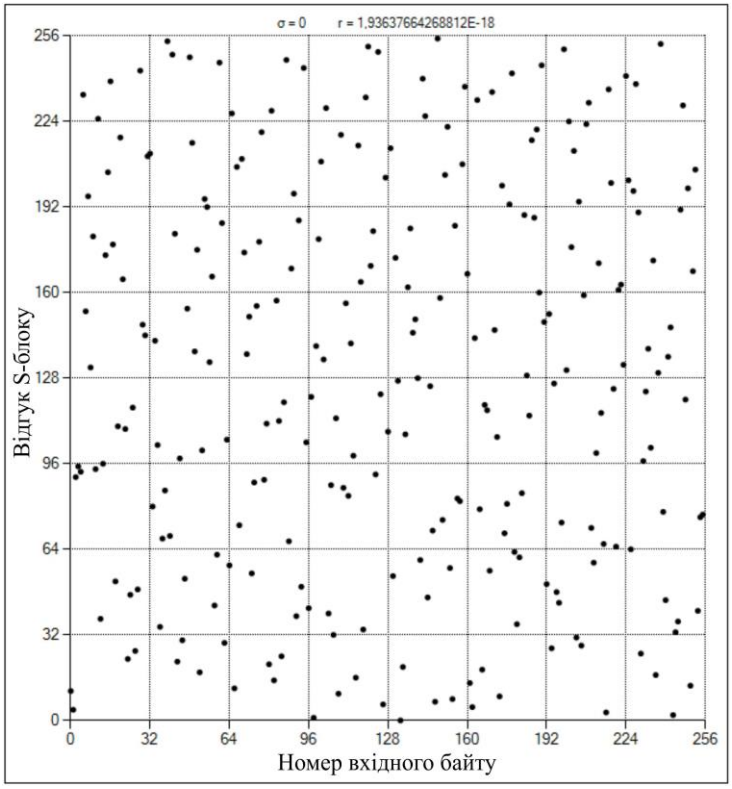


Рисунок 3.38 – Діаграма розсіювання рівномірно щільного примітиву нелінійнох підстановки

3.6. Блок перетворення байтів

БПБ складається з сукупності n рівномірно щільних ПНП і є основним вузлом запропонованого шифру. На рис.3.39 показана структурна схеми БПБ, до складу якої входять чотири ПНП, позначені як S-box$_i$, $i=\overline{0,3}$, типу «байт в байт».

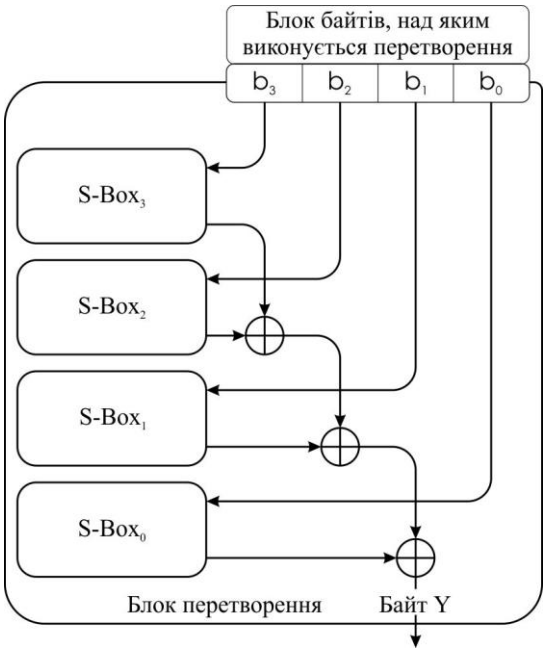


Рисунок 3.39 – Структурна схема чотирьохбайтного блоку перетворення

Узагальнена формула перетворення байтів b_i, $i=\overline{0,n-1}$, реалізуєма БПБ, така

$$y=S_{n-1}(b_{n-1})\oplus S_{n-2}(b_{n-2})\oplus\ldots\oplus S_0(b_0),$$

де $S_i(b_i)$ – відгук і-го S-блоку на вхідний байт b_i; $\oplus$ – оператор порозрядного складання по модулю 2 (операція XOR).

Вихідні байти Y БПБ (рис.3.39) можуть бути використані для поточного шифрування послідовностей байтів T відкритого тексту

$$C_i=T_i\oplus Y_i,\qquad i=0,1,\ldots \tag{3.8}$$

Співвідношення (3.8) передбачає, що для кожного i виконується нерівність $Y_{i+1}\neq Y_i$.

Для зламу послідовності C_i третьою стороною необхідно знати:

1) Розмірність m вектору байтів $B=\{b_i\}$, над яким виконуються перетворення;
2) Безпосередньо байти b_i, $i=\overline{0,m-1}$;
3) Стохастичну таблицю S-блоків.

Відзначимо що розмірність вектору n і сам вектор байтів B, можуть бути відкритими. Якщо таблиці S-блоків закриті, то шифрування (3.8) стає незламним. Насправді, станом на 2016 рік будь-яка атака ключем довжиною 128 біт (16 байт) по методу його послідовного перебору неможлива. Тим більше безглуздим є відтворення «всліпу» 256-байтного секретного ключа, не кажучи вже про те, що як в блоках перетворення, так і в шифраторах на їх основі, таких S-блоків може бути декілька, що значно знижує вірогідність зламу шифру.

Байт-орієнтовне потокове шифрування на основі рівномірно щільних блоків нелінійної підстановки засновано на використанні технічного рішення, відображеного на рис. 3.39 і 3.40.

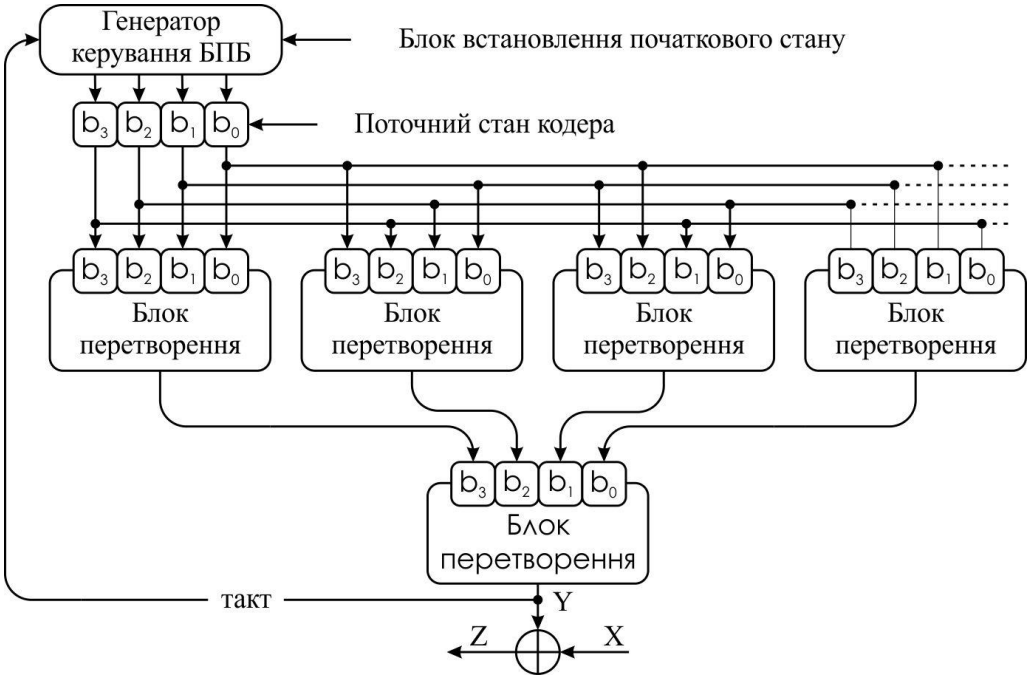


Рисунок 3.40 – Структура шифру на основі БПБ

Програмні цикли RSB (Round-Step-Block) шифру виконують такі функції: зовнішній цикл (i) задає крок шифрування; внутрішній цикл (i) визначає r-раундове шифрування. Керування примітивами здійснюється змістом раундових ключів RK (як базових, так і блокових), структура яких має вигляд (рис. 3.41).

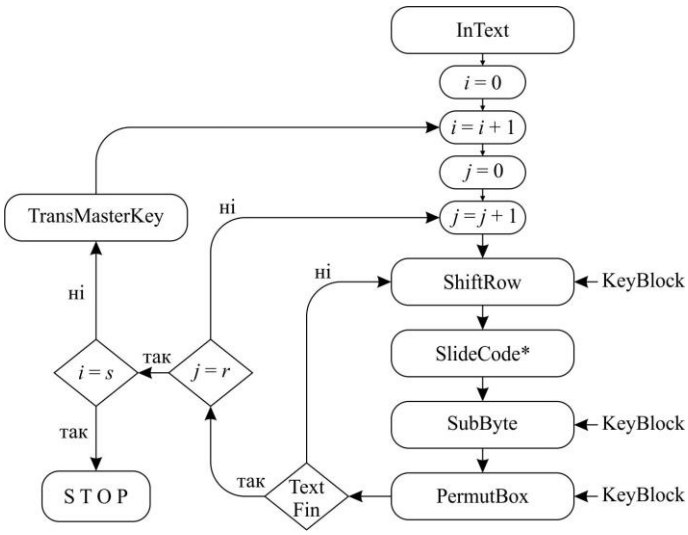


Рисунок 3.41 – Схема шифрування RSB-шифру

Відбувається динамічна модифікація SBox в блоках перетворення ключем шифрування.

3.7. Криптографічний протокол обміну даними

В основу протоколу містяться правила, регламентуючі використання криптографічних перетворень і алгоритмів в інформаційних процесах. Узагальнена схема побудування криптографічного протоколу зашифрування даних наведена на рис.3.42.

Протокол зашифрування вирішує наступні задачі:

- на етапі 1 формується блок інформаційних байтів;
- на етапі 2 доповнюється секретним кодом автентифікації (в авіаційній термінології – індивідуальним кодом «свій-чужий»), спільно утворюється інформаційний пакет (ІП);

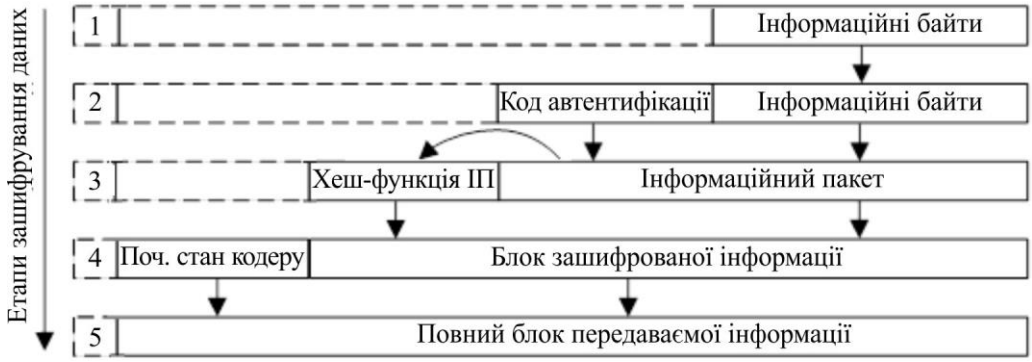


Рисунок 3.42 – Структурно-логічна схема протоколу зашифрування даних

- на етапі 3 розраховується хеш-функція (ХФ) інформаційного пакету;
- на етапі 4 об’єднаний блок ХФ + ІП зазнає криптографічного перетворення БПБ-шифром. До сформованого блоку зашифрованох інформації (БЗІ) приєднується блок початкового стану кодера (ПСК);
- на етапі 5 формується повний блок передачі інформації (БПІ) споживачу і, тим самим, завершується протокол зашифрування даних.

На приймаючій стороні етапи розшифрування даних виконуються в послідовності, зворотній послідовності етапам зашифрування (рис.3.43).

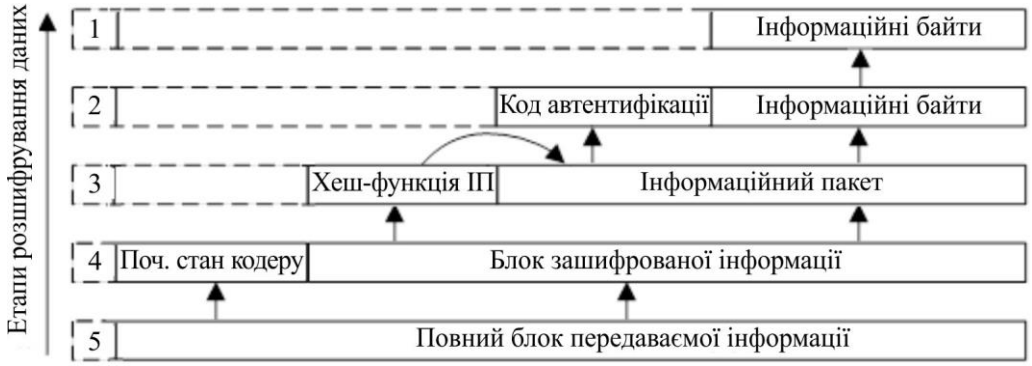


Рисунок 3.43 – Структурно-логічна схема протоколу розшифрування даних

Протоколом розшифрування даних виконуються наступні перетворення:

- на етапі 1 в процесорі приймача розміщується повний блок прийнятої інформації;
- на етапі 2 виділяються блоки ПСК і БЗІ;

- на етапі 3 за допомогою блоку ПСК запускається процес розшифрування даних по схемі, в результаті якого з БЗІ виділяють ХФ і ІП. Потім розраховується ХФ розшифрованого ІП. Якщо розрахована ХФ відрізняється від прийнятої, то ІП ігнорується. У випадку збігу хеш-значень процес продовжується;
- на етапі 4 з ІП вилучаються код автентифікації і файл інформаційних байтів;
- на етапі 5 розміщуються в регістрі інформаційні байти. На цьому завершується протокол розшифрування даних.

Роль синхронізуючого елементу в протоколах обміну даних виконує вміст блоку установки початкового стану генератору управління БПБ, яке передається по каналу зв’язку в відкритому вигляді, що не порушує секретність передачі даних, так як третя сторона не в змозі відтворити розшифровуючу гаму, оскільки для нього (противника) залишаються закритими S-блоки шифраторів.

3.8. Висновки до третього розділу

Запропоновані методи шифрування базуються на криптографічних примітивах лінійної і нелінійної заміни. При чому лінійні перетворення перемішування і ковзного кодування доповнюють множення матриць. Добуток матриць в тривимірному просторі має додаткові властивості порівняно з двовимірним добутком матриць. Наприклад, має значення напрямок множення. Всі ці параметри визначаються ключем шифрування. До нелінійних відносяться заміна за допомогою SBox 3D. Така таблиця складається з інших таблиць, які містять байти замін. Таким чином нелінійна заміна відбувається на декількох рівнях, через що кожен наступний блок при шифруванні використовує нову таблицю замін.

Особливість тривимірних криптографічних примітивів полягає в тому, що перетворення у структурованих даних відбуваються на декількох рівнях. Тобто на рівні байт, на рівні блоків з байт, на рівні структури з блоків і т.д.

Основні наукові результати отримані у даному розділі опубліковані у працях автора [2,3,5,7,8,9,71-79].

РОЗДІЛ 4
ПОБУДОВА ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ 3D ШИФРІВ

4.1. Методика проведення експериментів.

Для експерементальних досліджень було створено декілька програмних засобів. Перший реалізує популярні шифри, які використовуються компанією Майкрософт в своїй технології .Net (рис.4.1). Другий програмний засіб реалізує графічну оболонку, за допомогою якої можна керувати новою версією тестів Nist Sts (рис.4.2). Третя програма містить усі тести Dieharder (рис.4.3).

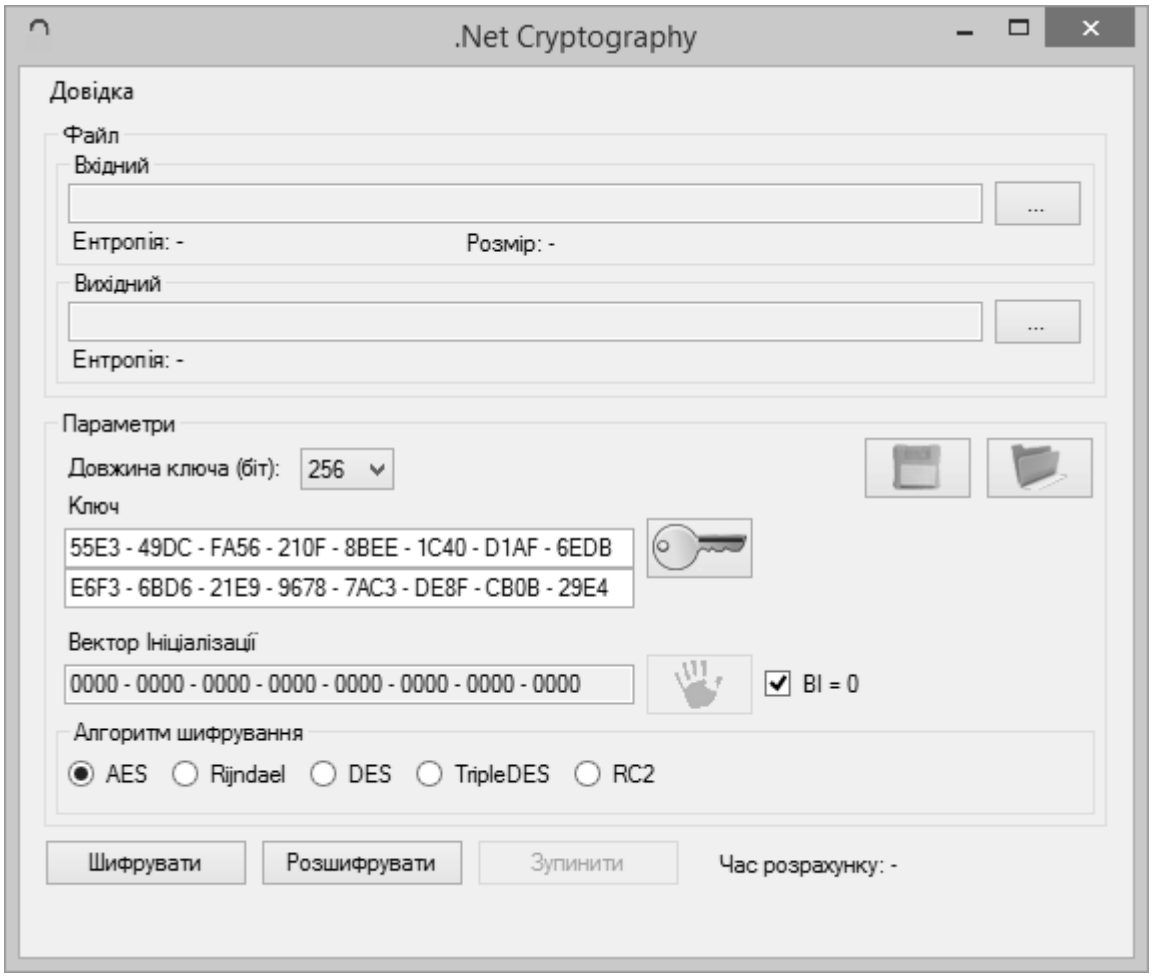


Рисунок 4.1 – Інтерфейс програмного засобу, що містить популярні блочні шифри, які використовуються в .Net Framework

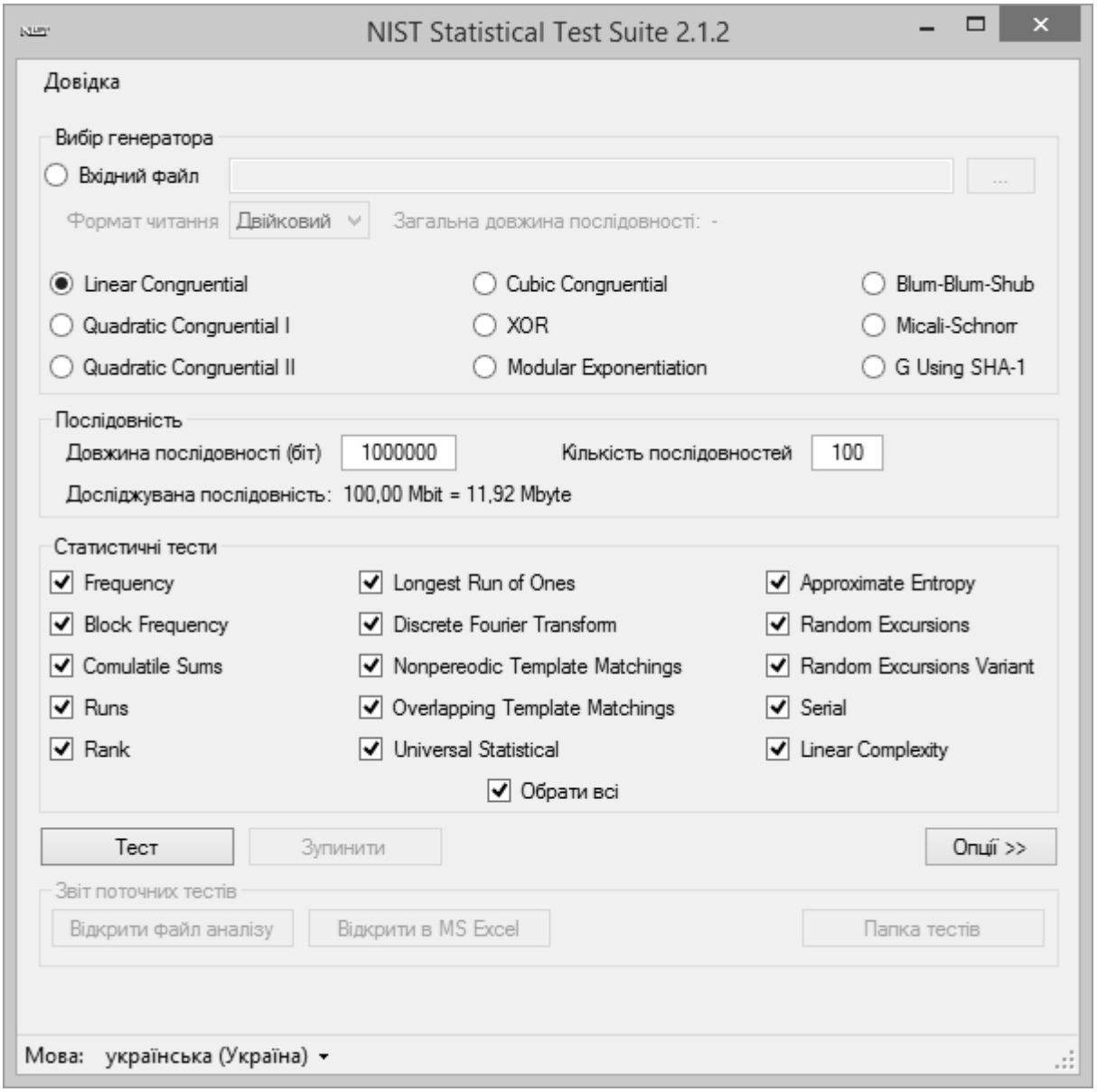


Рисунок 4.2 – Інтерфейс програмного засобу? що містить генератори псевдовипадкових послідовностей і усі статистичні тести Nist STS, для аналізу шифрограм

Функціональні можливості програми ".Net Cryptography". Платформа .Net Framework містить класи, які реалізують алгоритми шифрування з секретним ключем [47]: AES, DES, RC2, Rijndael, TripleDES. Ключ можна задавати довжиною 128, 198 і 256 біт. Можливо задавати вектор ініціалізації (BI) довжиною 128 біт, за замовчуванням BI не використовується (містить нульові значення, які не впливають на шифрування). Програма «.Net

Cryptography» розроблена для порівняння розробленого шифратора на основі 3D перетворень з шифрами AES, DES, RC2, Rijndael, TripleDES.

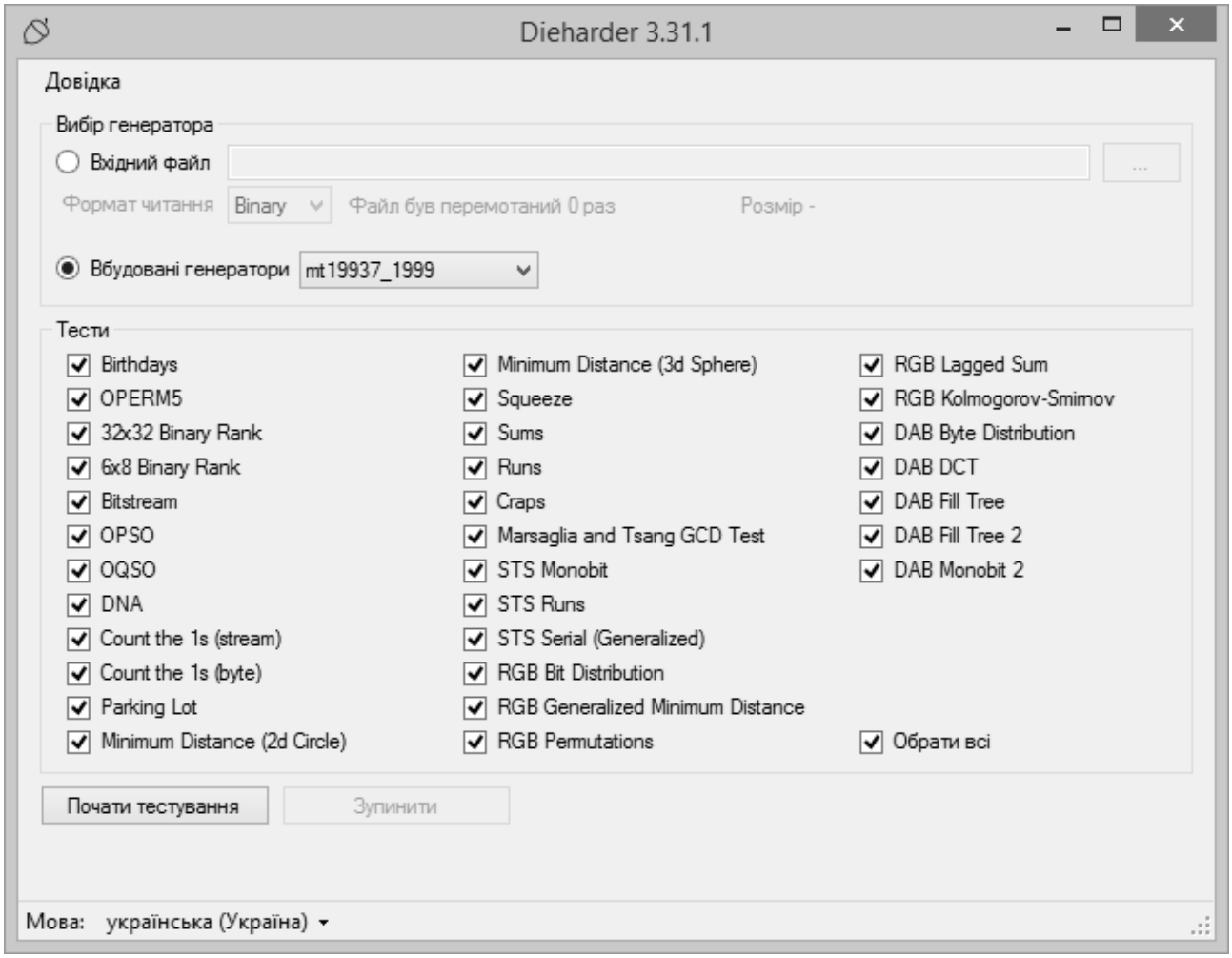


Рисунок 4.3 – Інтерфейс програмного засобу, що містить генератори псевдовипадкових послідовностей і усі статистичні тести Dieharder, для аналізу шифрограм

Функціональні можливості програми “NIST Statistical Test Suite 2.1.2”. Пакет статистичних тестів, розроблений Лабораторією інформаційних технологій (англ Information Technology Laboratory) в Національному інституті стандартів і технологій (NIST). В його склад входить 15 статистичних тестів, задачею яких є визначення міри випадковості двійкових послідовностей, що породжуються генераторами випадкових чисел. Ці тести основані тільки на статистичних властивостях, які притаманні тільки випадковим послідовностям.

Програма «NIST Statistical Test Suite 2.1.2» містить останню версію тестів NIST STS [12-14] від 9 липня 2014 р. Крім статистичних тестів, програма «NIST Statistical Test Suite 2.1.2» містить дев’ять генераторів псевдовипадкових послідовностей, які також описані в SP800-22rev1a [80] Програма «NIST Statistical Test Suite 2.1.2» зареєстрована в НАУ і захищена авторським свідоцтвом №59262.

Функціональні можливості програми “Dieharder 3.31.1”. Тести DieHarder це набір статистичних тестів, для вимірювання якості множини випадкових чисел. Тести були розроблені Джорджем Магсаглією. Програма «Dieharder 3.31.1» використовує останню версію тестів [81] які датовані 2015 роком: Крім статистичних тестів, програма «Dieharder 3.31.1» містить більше 70 генераторів псевдовипадкових послідовностей, які можна порівнювати з генераторами псевдовипадкових послідовностей розроблених на базі 3D перетворень.

Методика досліджень передбачала аналіз трьох типів файлів: текстовий (*.txt), графічний (*.jpg або *.png) і пустий файл. Текстовий файл має найнижчу ентропію, оскільки містить невеликий діапазон знаків з ASCII таблиці кодів (або іншої таблиці кодів). Графічний файл зазвичай має велику ентропію. Пустий файл це файл який заповнено байтами з 00 значеннями (в таблиці ASCII кодів нульовим байтом позначається відсутність даних). Фактично, за допомогою пустого файлу можливо подивитись наскільки добре шифратор працює як генератор псевдовипадкових послідовностей.

4.2. Дані для експериментів.

Для порівняльного аналізу використані два текстових файли, один з яких міститить Оксфордський словник англійської мови, інший – словник Даля. Характеристики досліджуваних текстових файлів наведені в табл.4.1.

Таблиця 4.1

Характеристики досліджуваних текстових файлів

Позначення файлу	Назва файлу	Розмір файлу	Ентропія	Найчастіші символи
A	Oxford dictionary of current English.txt	7 Мб	H = 4,8233…	Пробіл [32], ‘o’ [111]
B	Толковый словарь Даля .txt	17 Мб	H = 4,8788…	Пробіл [32], ‘O’ [174]

Статистичні портрети (рис.4.4 і рис.4.5) і значення ентропії отримані за допомогою розробленої програми «GISTO-H», яка зареєстрована в НАУ і захищена авторським свідоцтвом №52171.

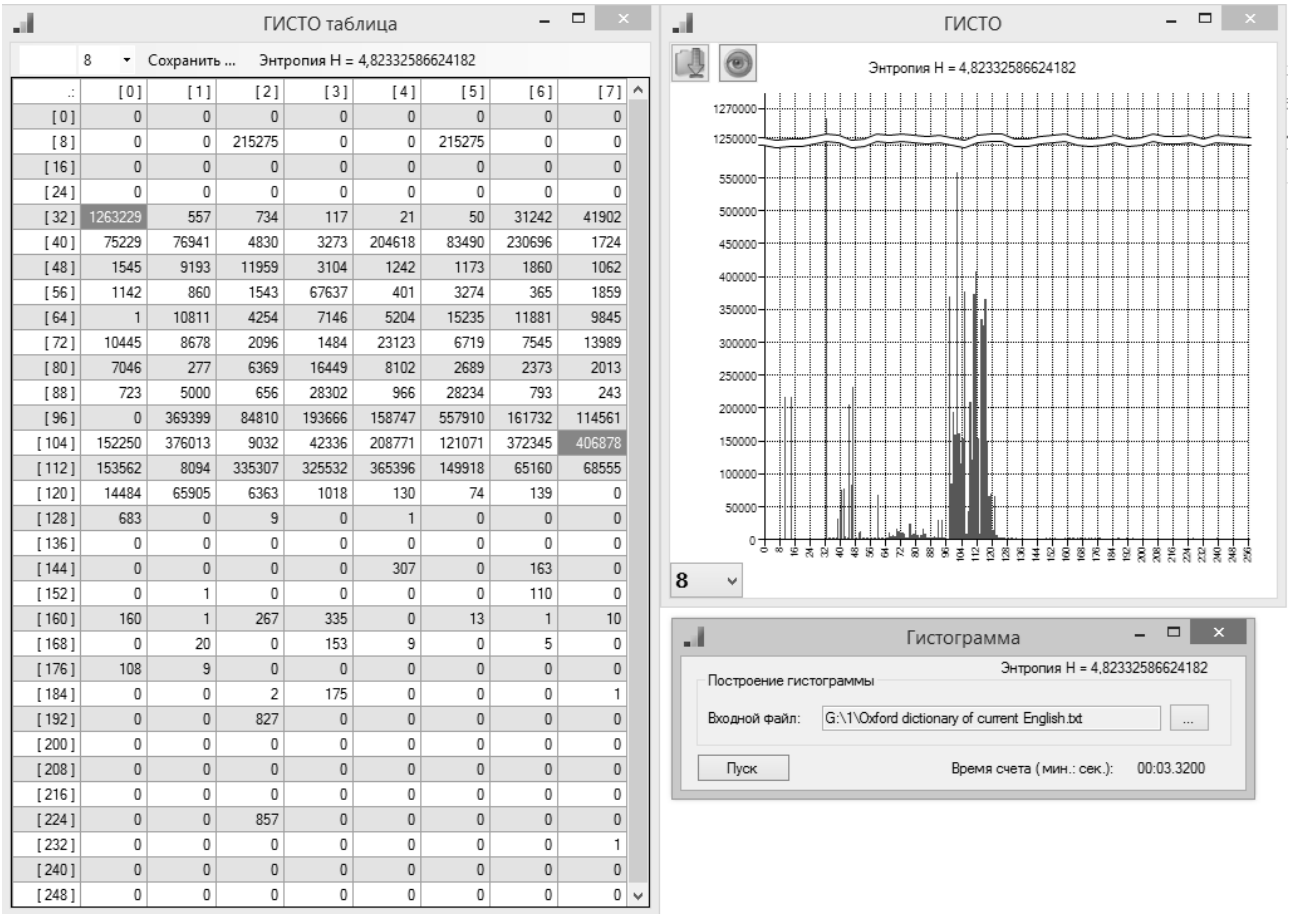
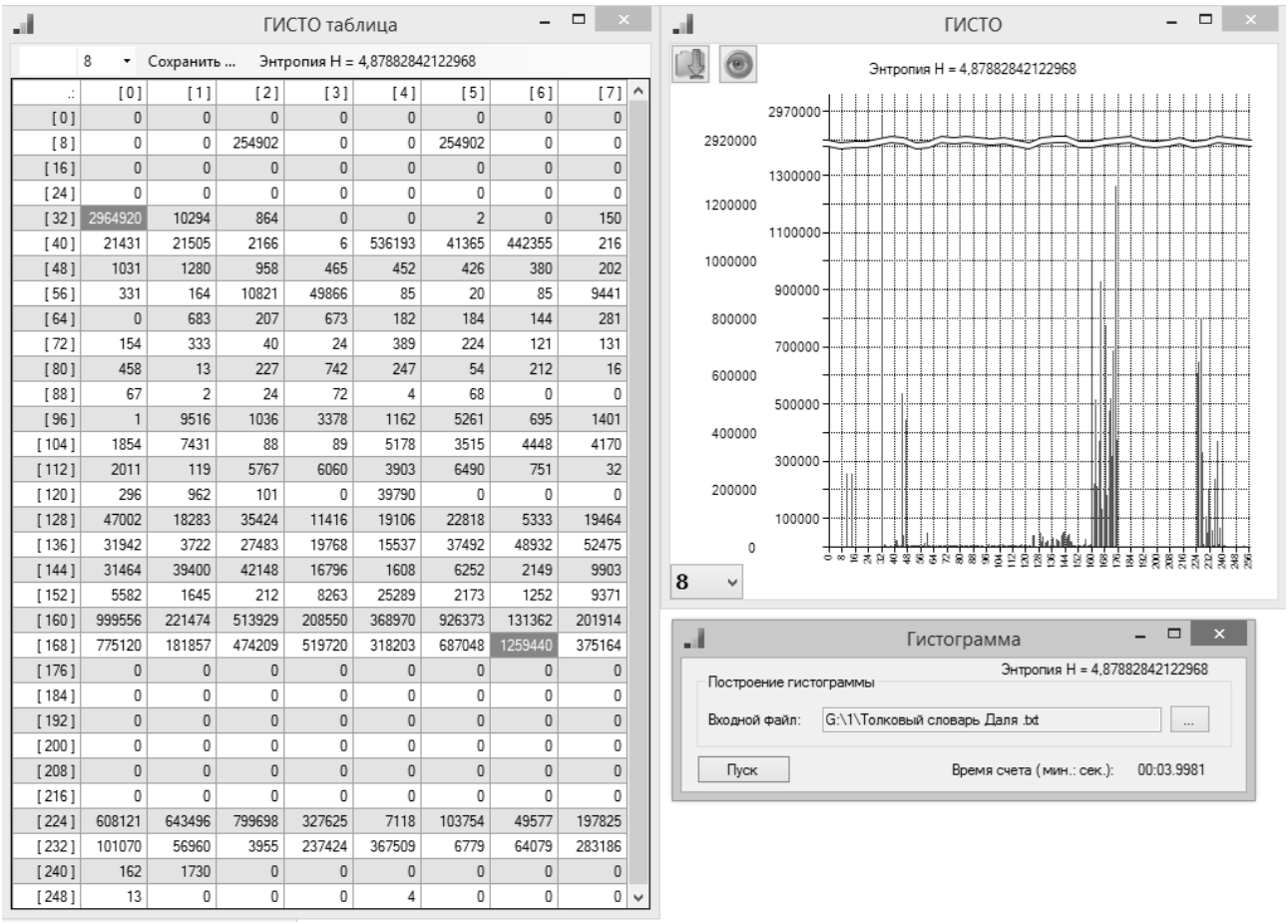


Рисунок 4.4 – Статистичний портрет файлу «Oxford dictionary of current English.txt»



.:	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
[0]	0	0	0	0	0	0	0	0
[8]	0	0	254902	0	0	254902	0	0
[16]	0	0	0	0	0	0	0	0
[24]	0	0	0	0	0	0	0	0
[32]	2964920	10294	864	0	0	2	0	150
[40]	21431	21505	2166	6	536193	41365	442355	216
[48]	1031	1280	958	465	452	426	380	202
[56]	331	164	10821	49866	85	20	85	9441
[64]	0	683	207	673	182	184	144	281
[72]	154	333	40	24	389	224	121	131
[80]	458	13	227	742	247	54	212	16
[88]	67	2	24	72	4	68	0	0
[96]	1	9516	1036	3378	1162	5261	695	1401
[104]	1854	7431	88	89	5178	3515	4448	4170
[112]	2011	119	5767	6060	3903	6490	751	32
[120]	296	962	101	0	39790	0	0	0
[128]	47002	18283	35424	11416	19106	22818	5333	19464
[136]	31942	3722	27483	19768	15537	37492	48932	52475
[144]	31464	39400	42148	16796	1608	6252	2149	9903
[152]	5582	1645	212	8263	25289	2173	1252	9371
[160]	999556	221474	513929	208550	368970	926373	131362	201914
[168]	775120	181857	474209	519720	318203	687048	1259440	375164
[176]	0	0	0	0	0	0	0	0
[184]	0	0	0	0	0	0	0	0
[192]	0	0	0	0	0	0	0	0
[200]	0	0	0	0	0	0	0	0
[208]	0	0	0	0	0	0	0	0
[216]	0	0	0	0	0	0	0	0
[224]	608121	643496	799698	327625	7118	103754	49577	197825
[232]	101070	56960	3955	237424	367509	6779	64079	283186
[240]	162	1730	0	0	0	0	0	0
[248]	13	0	0	0	4	0	0	0

Рисунок 4.5 – Статистичний портрет файлу «Толковый словарь Даля.txt»

Як видно на рисунках 4.4 і 4.5 у обох текстових файлів ентропія приблизно однакова. Файл в якому міститься тільки латинські символи має ентропію H = 4,82332586624182, файл що містить тільки символи кирилиці має ентропію H = 4,87882842122968. Різниця у другому знаку після зап’ятої принципова, але не суттєва. Цю різницю можна пояснити тим, що довжина алфавітів різна (26 в словнику Оксфорда і 32 в словнику Даля, знаки пунктуації однакові в обох словниках).

Для порівняльного аналізу використані два графічних файла. Формату *.png, він обраний замість *.jpg бо містить прозорі кольори. Другий файл буде у форматі *.bmp. Перший файл меншого розміру за рахунок стискання і коли

його зашифрують, то вже стає неможливо побачити графічну інформацію, бо усі службові данні, які потрібні для розпакування зображення, зашифровуються. Другий формат графічного файлу містить просторовий запис даних, в ньому відсутнє стискання, тому після зашифрування можна буде подивитись на якісні зміни графічного файлу. Характеристики досліджуваних графічних файлів наведено в табл.4.2.

Таблиця 4.2

Характеристики досліджуваних графічних файлів

Позначення файлу	Назва файл	Розмір файлу	Ентропія	Особливості
C	NAU logo.png	14 Мб	H = 7,9922…	10000x7893 пікселів
D	NAU logo.bmp	20 Мб	H = 5,0091…	3000x2367 пікселів

Дослідження проведені аналогічні як і для текстових файлів показали, що при зміні розмірів графічного файлу на декілька порядків значення ентропії змінюється в другому знаку після коми. Для дослідження були обрані файли більшого розміру, а саме файл «NAU logo.png» розміром 10000x7893 пікселів і 300dpi, файл «NAU logo.bmp» розміром 3000x2367 пікселів і палітрою 24 біта. Ентропія першого файлу H = 7,99222634897598, другого файлу H = 5,00919347830266. Зі значення ентропію зразу видно, що файл зі стисканням (графічний формат, що використовує архівацію) має майже ідеальну ентропію (максимально можлива ентропія H = 8,0), з другого боку графічний файл який просторово описує дані має ентропію наближену до ентропії текстового файлу (хоча і більшу). Гістограми статистичних портретів обох файлів зображено на рис.4.6.

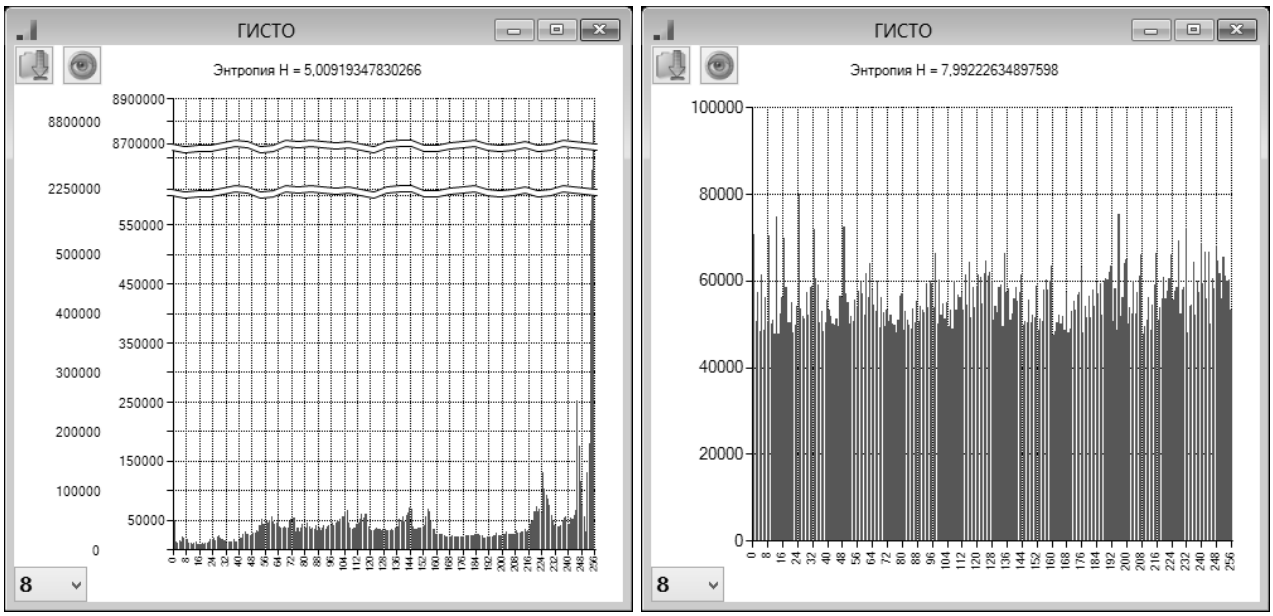


Рисунок 4.6 – Гістограми статистичних портретів *.bmp (зліва) і *.png (справа) файлів

В якості пустого файлу створено файл розміром 100Мбіт (12,5Мб), оскільки саме такий розмір досліджуємого файлу за замовчуванням пропонує Nist при аналізі статистичних властивостей. Створюється два файли, один заповнюється байтами зі значенням «0x00» (це 00000000 в бітовому вигляді) який позначаэться файл «E», другий файл заповнюється значеннями «0xFF» (це 11111111 в бітовому вигляді), який позначаэться файл «F». Відповідно ентропія для файлів «Пустий файл 0x00» і «Пустий файл 0xFF» буде дорівнювати H = 0,0… рис.4.7.

Методика проведення експериментів. За допомогою програми «Net Cryptography» зашифровуються всі зазначені вище файли різними шифраторами. Отримані шифрограми порівнюються з шифрограмами шифраторів «RSB-64-3D» (рис.4.8) і «M3DCrypt». Порівняння виконується за допомогою програм по статистичному аналізу шифрограм «NIST Statistical Test Suite 2.1.2», «Dieharder 3.31.1», ентропія знаходиться за допопогою програми «ГИСТО».

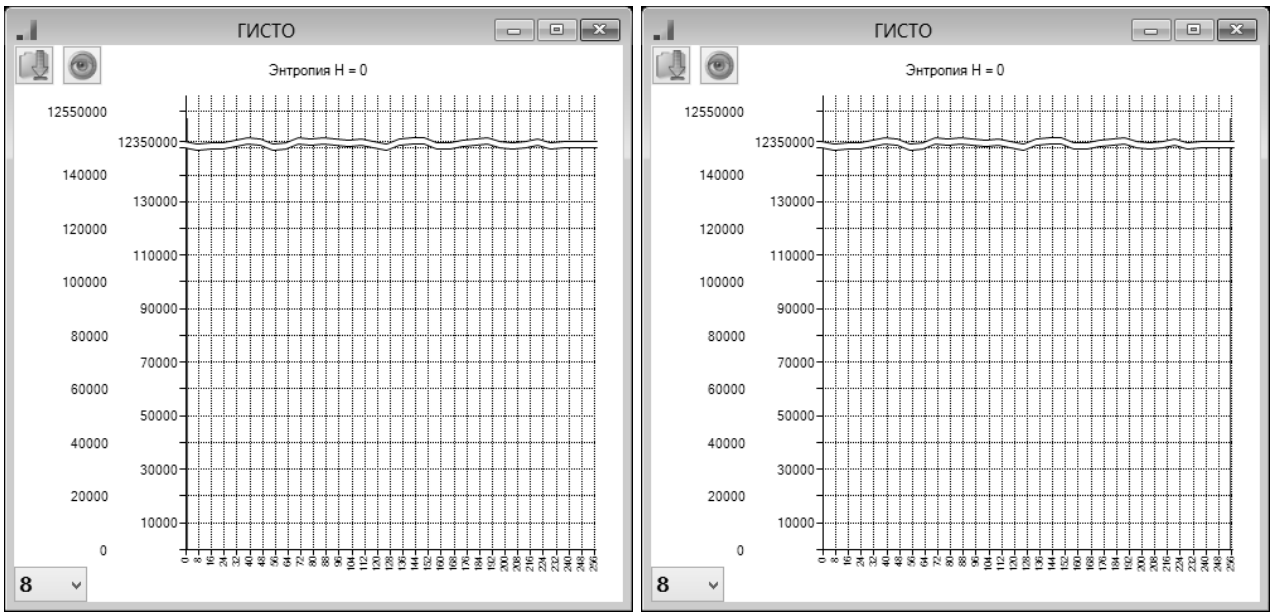


Рисунок 4.7 – Гістограми статистичних портретів «Пустий файл 0x00» (зліва) і «Пустий файл 0xFF» (справа) файлів

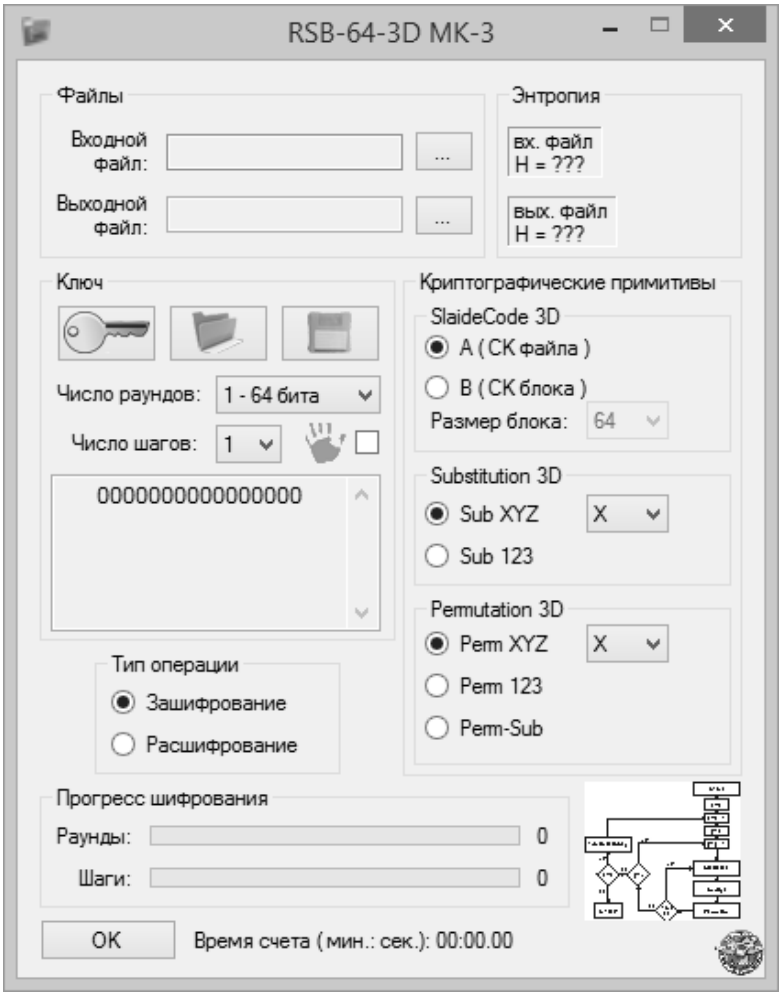


Рисунок 4.8 – Графічна оболонка шифратора RSB-64-3D

Під час експериментів використовується ключ шифрування довжиною 256 біт:

2DCB - 5624 - 7AF6 - 4A07 - D06D - 71EF - 2C6A - E9DF

37F1 - 7A2B - 6A01 - 6D75 - 99BC - 70AC - D618 - 27FF

В табл.4.4 наведено значення ентропій для різних шифрограм, які отримані з файлів від A до F.

Таблиця 4.4

Ентропії шифрограм

Файл	AES	3DES	RC2	RSB-64-3D	M3DCrypt
A	7,99997…	7,99997…	7,99998…	7,99997…	7,99997…
B	7,999990…	7,99998…	7,99998…	7,999990…	7,99998…
C	7,99998…	7,99998…	7,99998…	7,99998…	7,99998…
D	7,999991…	7,999992…	7,999991…	7,99990…	7,999991…
E	7,99998…	7,99998…	7,99998…	7,9993…	7,99998…
F	7,99998…	7,99998…	7,99998…	7,9996…	7,99998…

У табл.4.5 наведено різниці ентропій для шифрограм і вхідного файлу.

Таблиця.4.5

Різниця ентропій шифрограми і вхідного файлу

	AES	3DES	RC2	RSB-64-3D	M3DCrypt
A	3,1766488085	3,1766504387	3,1766552069	3,1766490331	3,1766513401
B	3,1211620846	3,1211613968	3,1211598510	3,1211617610	3,1211599211
C	0,0077615250	0,0077623574	0,0077611768	0,0077623858	0,0077602377
D	2,9907975592	2,9907986154	2,990797652	2,9907106095	2,9907984159
E	7,9999859287	7,9999865100	7,9999845266	7,9993225631	7,9999860300
F	7,9999874430	7,9999832727	7,9999858158	7,9996806320	7,9999860300

Для наочності дані наведені у табл.4.4. і табл.4.5. представлені у вигляді гістограм на рис.4.9 і рис.4.10.

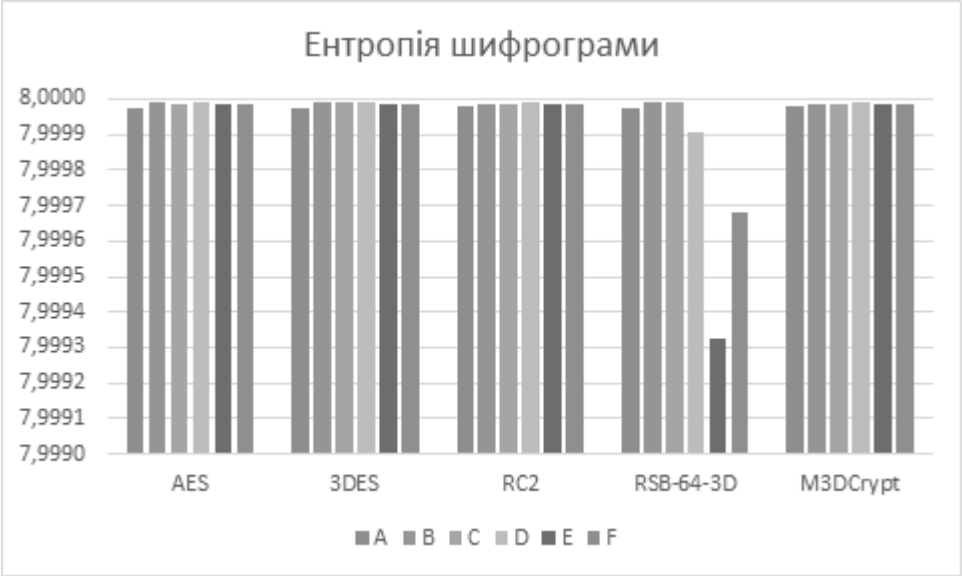


Рисунок 4.9 – Ентропії шифрограм

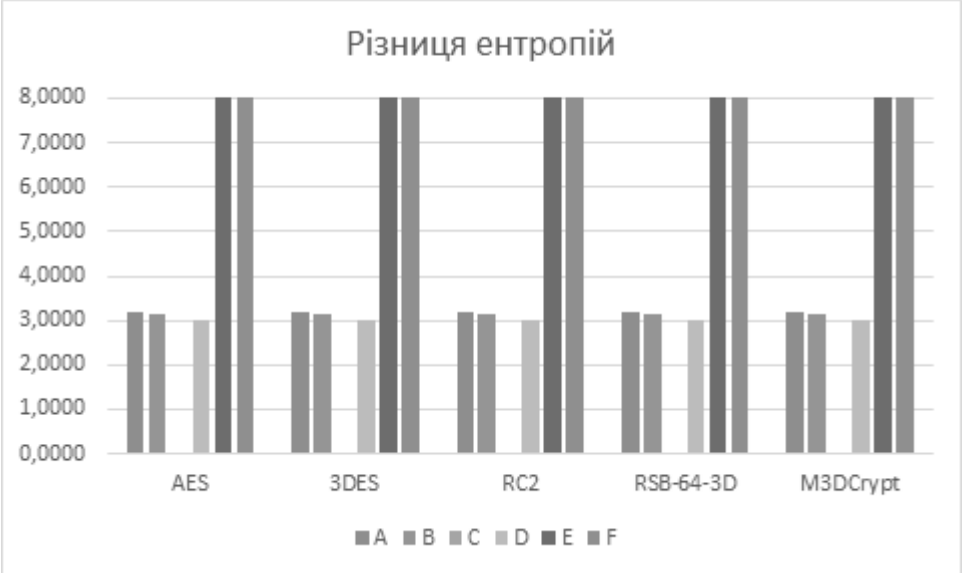


Рисунок 4.10 – Різниця ентропій шифрограми і вхідного файлу

4.3. Дослідження шифраторів.

Результати дослідження файлів за допомогою тесті Nist STS наведено на рис.4.11 і рис.4.12.

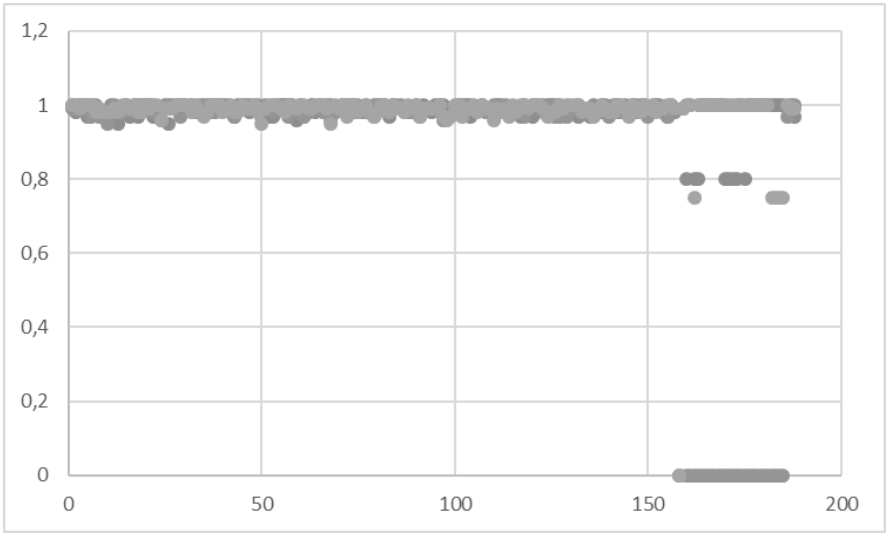


Рисунок 4.11 – Результати тестування за допомогою статистичних тестів Nist Sts шифрів AES256, 3DES, RC2

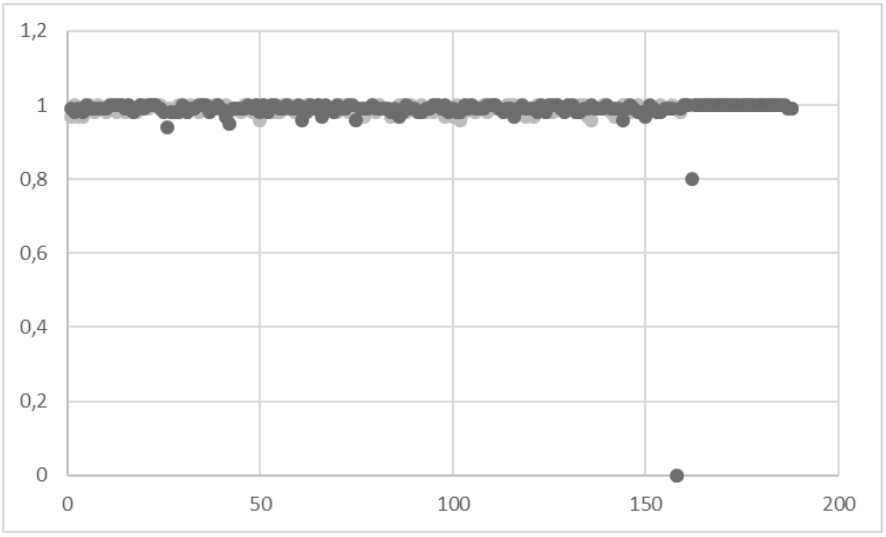


Рисунок 4.12 – Результати тестування за допомогою статистичних тестів Nist Sts шифрів RSB-64-3D і M3DCrypt

Результати дослідження файлів за допомогою тестів Dieharder наведено на рис.4.13, для наочності у вигляді графуку результати наведені на рис.4.14.

Рузультати відображають усереднину статистику для розроблених шифрів в порівнянні з шифрами AES256, 3DES, RC2 та генераторами псевдовипадкових чисел, що використовуються для порівняльного аналізу в Dieharder.

Назва тесту	ntup	P - value	Оцінка
Birthdays	0	0,92475787	Зараховано
OPERM5	0	0,43116433	Зараховано
32x32 Binary Rank	0	0,17346659	Зараховано
6x8 Binary Rank	0	0,59645826	Зараховано
Bitstream	0	0,47561663	Зараховано
OPSO	0	0,94397523	Зараховано
OQSO	0	0,89526067	Зараховано
DNA	0	0,55781442	Зараховано
Count the 1s (stream)	0	0,95812837	Зараховано
Count the 1s (byte)	0	0,63469287	Зараховано
Parking Lot	0	0,5293412	Зараховано
Minimum Distance (2d Circle)	2	0,54586473	Зараховано
Minimum Distance (3d Sphere)	3	0,11395833	Зараховано
Squeeze	0	0,06980951	Зараховано
Sums	0	0,01592635	Зараховано
Runs	0	0,15666244	Зараховано
Runs	0	0,62954573	Зараховано
Craps	0	0,47382495	Зараховано
Craps	0	0,75270773	Зараховано
Marsaglia and Tsang GCD Test	0	0,14476831	Зараховано
Marsaglia and Tsang GCD Test	0	0,79527376	Зараховано
STS Monobit	1	0,0485777	Зараховано
STS Runs	2	0,5813378	Зараховано
STS Serial (Generalized)	1	0,16496516	Зараховано

Рисунок 4.13 – Результати тестування за допомогою генераторів ПВЧ Dieharder шифрів RSB-64-3D і M3DCrypt

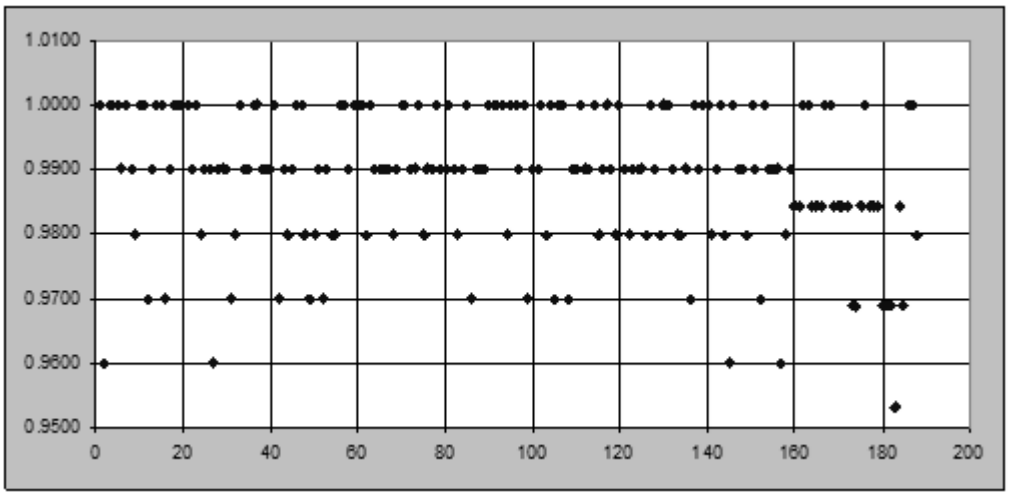


Рисунок 4.14 – Результати тестування за генераторів ВПЧ Dieharder шифрів RSB-64-3D і M3DCrypt разом з тестами AES256, 3DES, RC2

Аналіз цих результатів показує, що всі ці тести проходять.

4.4. Оцінки теоретичної стійкості шифрів на основі 3D перетворень

Оцінки стійкості отримані виходячи з основних положень роботи [82].

Опишемо математично БШ RSB-64-3D. Розглянемо r-раундовий блоковий шифр $\Im$ із множиною відкритих (шифрованих) повідомлень $V_n = \{0,1\}^n$, множиною раундових ключів $K = V_n$ та сімейством шифруючих перетворень

$$F_k = f_{r,k_r} \circ \ldots \circ f_{1,k_1},\ k = (k_1,\ldots,k_r) \in K^r, \tag{4.1}$$

де t, p', r' – натуральні числа, $p = 4p'$, $p'' = p/2$, $r = 2r'+1$, $t' = 2t$, $n = t'p''$.

Раундова функція $f_{i,k}$, для будь-яких $x \in V_n$, $k \in K$, $i \in \overline{1,r}$ описується наступним чином:

$$f_{i,k} = \begin{cases} \varphi(x \oplus k), \text{якщо} & i < r \\ s(x \oplus k), \text{якщо} & i = r \end{cases}. \tag{4.2}$$

Підстановки φ і s визначаються за формулами:

$$\varphi(x) = s(x)M,\ x \in V_n, \tag{4.3}$$

$$s(x) = (s_{p''-1}(x_{p''-1}),\ldots,s_0(x_0)),\ x = (x_{p''-1},\ldots,x_0), \tag{4.4}$$

де $x_j \in V_t$, s_j – підстановка на множині V_t, $j \in \overline{0, p^n - 1}$.

M – оборотна $p \times p$ – матриця над полем $GF(2^8)$, а множення $s(x)$ на M у виразі (4.3) виконується над цим полем у декілька етапів:

1) $s_j(x_j)$ ($j \in \overline{0, p^n - 1}$) розкладається на дві 8-бітні частини ($s_j(x_j) = (s_j(x_j)^{(1)}, s_j(x_j)^{(2)})$, $s_j(x_j)^{(1)}, s_j(x_j)^{(2)} \in V_t$).

2) Із частин $s_j(x_j)$ ($j \in \overline{0, p^n - 1}$) формується вектор y: $y = s_0(x_0^{(1)}) | s_0(x_0^{(2)}) | \ldots | s_{p''-1}(x_{p''-1}^{(1)}) | s_{p''-1}(x_{p''-1}^{(2)})$.

3) Виконується множення вектора y на M при ототожненні двійкових векторів y_j, $j \in \overline{0, p-1}$ ($y_j \in V_t$) із елементами матриці M: $\varphi(x) = y \cdot M$.

Символ $\oplus$ відповідає операції по координатного булевого додавання двійкових векторів.

Конкретним прикладом шифру $\Im$ є БШ RSB-64-3D. У цьому випадку $t = 8$, $p' = 4$, $r' = 3$ ($r' = 4$, $r' = 8$) – натуральні числа, $p = 4p' = 16$, $p'' = p/2 = 8$, $r = 2r' + 1 = 7$ ($r = 9$, $r = 17$), $t' = 2t = 16$, $n = t' p'' = 128$.

Використовується лиша одна таблиця замін на множині V_{16}, а матриця M має такий вигляд:

$$M = \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix} diag(D, D), \tag{4.5}$$

де I та 0 – відповідно одинична та нульова матриця порядку 8 на полем $GF(2^8)$, D – МДР матриця порядку 8 над цим полем.

Із робіти [82] слідує, що якщо БШ описується співвідношеннями (4.1)–(4.4), то для нього при будь-яких $x \in V_n$ виконуються наступні твердження:

якщо $i \equiv 1 (\mathrm{mod}\, 2)$, $i < r$, то

$$d_x^{(i)}(\omega_{i-1}, \omega_i) \le \left(\tilde{\Delta}_{\oplus} \right)^{wt(\omega_i M^{-1})}, \tag{4.6}$$

$$l^{(i)}(\omega_{i-1}, \omega_i) \le \left(\tilde{\Delta}_{\oplus} \right)^{wt(\omega_i M)}; \tag{4.7}$$

якщо $i = r$, то

$$d_x^{(i)}(\omega_{i-1},\omega_i)\le\left(\tilde{\tilde{\Delta}}_{\oplus}\right)^{wt(\omega_i)}, \tag{4.8}$$

$$l^{(i)}(\omega_{i-1},\omega_i)\le\left(\tilde{\tilde{\Delta}}_{\oplus}\right)^{wt(\omega_i)}; \tag{4.9}$$

якщо $i<r$, то

$$wt\left(\omega_i M^{-1}\right)=wt(\omega_{i-1}), \tag{4.10}$$

$$wt(\omega_i M)=wt(\omega_{i-1}); \tag{4.11}$$

якщо $i=r$, то

$$wt(\omega_r)=wt(\omega_{r-1}), \tag{4.12}$$

$$wt(\omega_r)=wt(\omega_{r-1}). \tag{4.13}$$

На основі чого, згідно доведенню теореми для БШ, що описуються (4.1)−(4.4) виконується нерівності:

$$EDP(\Omega)\le\Delta_{\oplus}^{\,r'\lceil B_M/2\rceil+1}, \tag{4.14}$$

$$ELP(\Omega)\le\Delta_{\oplus}^{\,r'\lceil B_M/2\rceil+1}. \tag{4.15}$$

Оскільки для БШ RSB-64-3D $\Delta_{\oplus}=\Lambda_{\oplus}=2^{-14}$ та $\lceil B_M/2\rceil=5$ то на основі (4.14)-(4.15) були визначені верхні межі стійкості відносно методів ЛДК (табл. 4.6), що свідчать про практичну стійкість БШ M3DCrypt відносно методів ЛДК.

Тепер, опишемо математично БШ M3DCrypt. Розглянемо r-раундовий блоковий шифр $\tilde{\tilde{\Im}}$ із множиною відкритих (шифрованих) повідомлень $V_n=\{0,1\}^n$, множиною раундових ключів $K=V_{n+q}$ та сімейством шифруючих перетворень

$$F_k=f_{r,k_r}\circ\ldots\circ f_{1,k_1},\ k=(k_1,\ldots,k_r)\in K^r, \tag{4.16}$$

де t, p, r', q' − натуральні числа, $r=2r'+1$, $n=pt$, $q=pq'$ (параметр, що визначає довжину додаткового раундового ключа), $b=2^{q'}$ (параметр b визначає

кількість різних таблиць підстановок, що можуть використовуватись в заданому БШ).

Раундова функція $f_{i,k}$, для будь-яких $x \in V_n$, $k \in K$, $i \in \overline{1,r}$ описується наступним чином:

$$f_{i,k}(x) = \begin{cases} \varphi\left(x \oplus k^{(1)}, k^{(2)}\right), \text{якщо} & i < r \\ s\left(x \oplus k^{(1)}, k^{(2)}\right), \text{якщо} & i = r \end{cases}, \tag{4.17}$$

де $k^{(1)}$ та $k^{(2)}$ частини раундового ключа k ($k = \left(k^{(1)}, k^{(2)}\right)$, $k^{(1)} \in V_n$, $k^{(2)} \in V_q$).

Підстановки φ і s визначаються за формулами:

$$\varphi(x, y) = s(x, y)M,\ x \in V_n,\ y \in V_q, \tag{4.18}$$

$$s(x, y) = \left(s_{y_{p-1}}(x_{p-1}), \ldots, s_{y_0}(x_0)\right),\ x = (x_{p-1}, \ldots, x_0), y = (y_{p-1}, \ldots, y_0), \tag{4.19}$$

де $x_j \in V_t$, $y_j \in V_q$, s_{y_j} – підстановка на множині V_t (по індексу y_j обирається для використання одна таблиця підстановок із b можливих), $j \in \overline{0, p-1}$, M – оборотна $p \times p$ – матриця над полем $GF\left(2^t\right)$, а множення $s(x, y)$ на M у виразі (4.18) виконується над цим полем при ототожненні двійкових векторів $s_{y_j}(x_j)$ із його елементами. Символ $\oplus$ відповідає операції по координатного булевого додавання двійкових векторів.

Конкретним прикладом шифру $\tilde{\tilde{\mathfrak{I}}}$ є БШ M3DCrypt. У цьому випадку $t = 8$, $r' = 3$ ($r' = 4$, $r' = 9$), $p = 16$, $r = 2r' + 1 = 7$ ($r = 9$, $r = 19$), $n = pt = 128$, $b = 16$, $q' = 4$, $q = pq' = 64$. Використовується 16 таблиць на множині V_8, при чому вибір конкретної таблиці у кожному раунді залежить від раундового ключа, а матриця M представляється за формулою 4.5.

Згідно доведенню формул [20] для БШ, що описуються (4.16)–(4.19) виконується нерівності:

$$EDP(\Omega) \le \tilde{\tilde{\Delta}}_{\oplus}^{r' B_M + 1}, \tag{4.20}$$

$$ELP(\Omega) \le \tilde{\tilde{\Delta}}_{\oplus}^{r'B_M+1}. \qquad (4.21)$$

Оскільки для БШ M3DCrypt виконуються $\tilde{\tilde{\Delta}}_{\oplus} = 2^{-7}$, $\tilde{\tilde{\Lambda}}_{\oplus} = 2^{-6,75}$ (параметри $\tilde{\tilde{\Delta}}_{\oplus}$ та $\tilde{\tilde{\Lambda}}_{\oplus}$ були відповідно розраховані за формулами (2.13)-(2.25) та (2.40)-(2.52) за усіма 16 таблиць замін) та $B_M = 9$ то на основі (4.20)-(4.21) були визначені верхні межі стійкості відносно методів ЛДК (табл. 4.7), що свідчать про практичну стійкість БШ M3DCrypt відносно методів ЛДК.

Таблиця 4.7

Аналітичні верхні межі стійкості відносно методів ЛДК шифрів

Розмір ключа K (біт)	RSB-64-3D		M3DCrypt	
	Диференціальний криптоаналіз	Лінійний криптоаналіз	Диференціальний криптоаналіз	Лінійний криптоаналіз
128	$EDP(\Omega) \le 2^{-193}$	$ELP(\Omega) \le 2^{-186}$	$EDP(\Omega) \le 2^{-185}$	$ELP(\Omega) \le 2^{-179}$
256	$EDP(\Omega) \le 2^{-257}$	$ELP(\Omega) \le 2^{-247}$	$EDP(\Omega) \le 2^{-263}$	$ELP(\Omega) \le 2^{-237}$

4.4. Висновки до четвертого розділу

Розроблені програмні засоби забезпечили експерементальне дослідження як відомих так і запропонованих шифрів.

Експерементальні дослідження розроблеих шифрів RSB-64-3D і M3DCrypt показали що вони за більшістью тестів мають такі самі властивості як і відомі шифри AES256, 3DES, RC2. Однак для деяких тестів запропоновані шифри мають переваги над відомими шифрами.

Отримані аналітичні верхні межі стійкості запропонованих шифрів RSB-64-3D, M3DCrypt, показали що ці шифри задовольняють вимогам що висуваються до сучасних шифрів.

Програмні засоби і способи криптографічного захисту інформації що використовувались для експерементальних досліджень мають свідоцтва і патенти [71-79].

Основні наукові результати отримані у даному розділі опубліковані у працях автора [3-7, 71-79].

ОСНОВНІ ВИСНОВКИ

У дисертаційній роботі на основі апарату 3D модулярної алгебри розв’язано актуальне науково-технічне завдання щодо розроблення методів побудови симетричних криптосистем (блокових та потокових шифрів) для підвищення ефективності захисту інформації.

Основні наукові та практичні результати дисертаційної роботи полягають у такому:

1. Проведено аналіз сучасних криптографічних методів захисту інформації за критеріями стійкості, швидкодії та необхідних ресурсів для розрахунків, що дало можливість виявити переваги за зазначеними критеріями шифрів на основі 2D перетворень над 1D, а також обґрунтувати необхідність розроблення нових методів за рахунок використання в шифраторах керованих 3D криптографічних перетворень.

2. Розроблені методи формування динамічно керованих примітивів лінійного розсіювання, нелінійної заміни та «ковзного кодування» на основі узагальнених перетворень Грея та матриць Галуа для тривимірного простору, що дало можливість запропонувати клас симетричних алгоритмів шифрування (потокових і блокових шифрів), орієнтованих на максимально можливе використання керованих 3D криптографічних примітивів. За рахунок того, що всі перетворення, які виконуються алгоритмом, стають залежними не лише від секретного ключа, але і від даних, що шифруються. Тим самим примітиви набувають властивість керованих криптоперетворень.

3. На основі запропонованих алгоритмічних рішень розроблено методи криптографічної обробки даних (блокового, потокового) тривимірного шифрування даних, що сприяло підвищенню стійкості, швидкодії і зменшенню необхідних ресурсів для забезпечення криптографічного захисту інформації за рахунок розроблених алгоритмічних методів побудови просторових кубічних матриць четвертого порядку, які використовуються для зберігання ключів і даних. В основу синтезу просторових матриць покладені

так звані узагальнені матриці Галуа. На відміну від відомих (класичних) матриць Галуа, які будуються виключно на основі примітивних незвідних поліномів, узагальнені матриці Галуа можуть бути побудовані за допомогою поліномів, які не обов'язково повинні бути примітивними, за умови, що примітивним є елемент, породжуючий матрицю Галуа, завдяки цьому значно поширюється множина просторових матриць Галуа.

4. Розроблено апаратно-програмне забезпечення і проведені лабораторні випробування засобів захисту командно-телеметричної та відеоінформації в каналах зв'язку наземного пункту керування з безпілотним літальним апаратом, це надало можливість суттєво збільшити як довжину періоду гамма-послідовності, так і швидкість криптографічних перетворень за рахунок запропонованого оригінального байт-орієнтовного потокового шифрування даних на основі рівномірно щільних блоків нелінійної підстановки.

5. Практичне значення отриманих результатів полягає в тому, що: а) запропоновано спосіб тривимірних перетворень блокових та потокових шифрів (БШ і ПШ) для побудови більш ефективних криптосистем; б) розроблені та впроваджені алгоритми симетричного блокового і 3DMatrix потокового шифрування інформації, які доведені до рівня програмної реалізації на мовах C# і C++, що дають можливість підвищити ефективність криптографічного захисту інформації; в) розроблено ПЗ на основі запропонованих шифрів, що сприяло проведенню експериментального дослідження запропонованих рішень для криптографічного захисту командно-телеметричної інформації в каналі зв'язку НПК-БПЛА; г) за результатами дисертаційних досліджень отримано дев'ять патентів України на «Спосіб криптографічного захисту інформації».

6. Результати дисертаційної роботи впроваджено у навчальному процесі кафедри електроніки Національного авіаційного університету (акт впровадження від 13.02.17 р.), кафедри виробництва приладів Національного технічного університету України «Київський політехнічний інститут імені

Ігоря Сікорського» (акт впровадження від 15.02.17 р.) та в науково-технічних розробках ТОВ «Агфар» (акт впровадження від 07.02.17 р.), ТОВ «Гратис, Лтд» (акт впровадження від 13.02.17 р.), що підтверджено відповідними актами впровадження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Shutko V. N.* Possibility of images recognition in navigation by artificial system / V.N. Shutko, O.M. Klyuchko, D.O. Navrotskyi // Methods and Systems of Navigation and Motion Control (MSNMC): 2014 IEEE 3rd International Conference. – 2014. – P. 165–169.

2. *Белецкий А. Я.* Синтез систем дискретных Уолша-подобных секвентных функций восьмого порядка / А. Я. Белецкий, Д. А. Навроцкий // Безпека інформації. – 2016. – Т. 22, № 2. – С. 163–174.

3. *Белецкий А. Я.* Алгоритм байт-ориентированного поточного шифрования на основе равномерно плотных блоков нелинейной подстановки // А. Я. Белецкий, Д. А. Навроцкий, А.И. Семенюк // Захист інформації. – 2016. – Т. 18, № 2. – С. 114–123.

4. *Навроцький Д.О.* Криптографічна система захисту радіоканалів БПЛА від несанкціонованого втручання / Д. О. Навроцький // Безпека інформації. – 2014. – Т. 20, № 3. – С. 248–252.

5. *Белецкий А.* Программно-моделирующий комплекс SCSPS алгоритма поточного шифрования / А. Белецкий, Д. Навроцкий, А. Семенюк // Захист інформації. – 2014. – Т. 16, № 2. – С. 113-121.

6. *Программно-моделирующий* комплекс ВРС алгоритма поточного шифрования и помехоустойчивого кодирования видеосигналов, передаваемых с борта БПЛА / [А. Я. Белецкий, А. В. Максименко, Д. А. Навроцкий та інші] // Захист інформації. – 2014. – Т. 16, № 3. – С. 184–191.

7. *Программно-моделирующий* комплекс криптографических AES-подобных примитивов нелинейной подстановки / [А. А. Белецкий, А. Я.

Белецкий, Д. А. Навроцкий, А.И. Семенюк] // Захист інформації. – 2014. – Т. 16, № 1. – С. 12–22.

8. *Примитивные* полиномы в криптографических приложениях / А. Я. Белецкий, Е. А. Белецкий, Р. Ю. Кандиба, Д. А. Навроцкий // Сучасний захист інформації. – 2011. – № 4. – С. 5–18.

9. *The* set of program models for ecological monitoring technical system based on principles of biophysics / [O. M. Klyuchko, V. N. Shutko, D. O. Navrotskyi, A. M. Mikolushko] // Електроніка та системи управління. – 2014. – № 4. – С. 135–142.

10. *Навроцький Д.О.* Дослідження результатів стеганографічного приховування повідомлень у файлах зображення // Д.О. Навроцький / Вісник Національного технічного університету України «Київський політехнічний інститут» – Серія «Радіотехніка. Радіоапаратобудування». – 2012. – № 50. – С.121–128.

11. *Двоичные* квазиэквидистантные и отраженные коды в смешанных системах счисления / [А. Я. Белецкий, Е. А. Белецкий, Р. Ю. Кандиба, Д. А. Навроцкий] // Вісник СумДУ. Серія «Технічні науки». – 2012. – № 1. – С. 42–58.

12. *Morris Dworkin.* Recommendation for Block Cipher Modes of Operation // Morris Dworkin / Natl. Inst. Stand. Technol. Spec. Publ. 800-38A. – 2001. – P.66.

13. *Morris Dworkin.* Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality / Morris Dworkin // Natl. Inst. Stand. Technol. Spec. Publ. 800-38C. – 2004. – P.25.

14. *Morris Dworkin.* Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC / Morris Dworkin // Natl. Inst. Stand. Technol. Spec. Publ. 800-38D. – 2007. – P.37.

15. Лужецький В. А. Аналіз алгоритмів симетричного блокового шифрування / В. А. Лужецький, А. В. Остапенко // Інформаційні технології та комп'ютерна інженерія - 2012 –№3 (25) – С. 55-65.

16. Скляров Д.В. Искусство защиты и взлома информации / Д.В. Скляров // СПб.: БХВ-Петербург. – 2005. – 288 с.

17. Бабаш А.В. Криптография / Бабаш А.В., Шанкин Г.П. – М.: СОЛОН-Р, 2002. – 512 с.

18. Cryptography Next Generation. [Електронний ресурс] – Режим доступу: https://msdn.microsoft.com/en-us/library/windows/desktop/aa376210(v=vs.85).aspx

19. Брассар Ж. Современная криптология. / Брассар Ж. – М.: Полимед, 1999. – 178 с.

20. Винокуров А. Сравнение российского стандарта шифрования, алгоритма ГОСТ 28147-89, и алгоритма Rijndael, выбранного в качестве нового стандарта шифрования США / Винокуров А., Применко Э. // Системы безопасности. – М.: Гротэк. – 2001. № № 1, 2.

21. ДСТУ 7624:2014. Інформаційні технології. Криптографічний захист інформації. Алгоритм симетричного блокового перетворення. [Текст]. – Введ. 01–07–2015. – К.: Мінекономрозвитку України, 2015.

22. Словарь криптографических терминов / Под ред. Б.А. Погорелова и В.Н. Сачкова. – М.: МЦНМО, 2006. – 94 с.

23. *Фергюсон Н.* Практическая криптография. – М.: Изд. дом «Вильямс», 2005. – 424 с.

24. *Смарт Н.* Криптография / Смарт Н. – М.: Техносфера, 2005. – 528 с.

25. *Рябко Б.Я.*, Фионов А.Н. Основы современной криптографии и стеганографии. – М.: Горячая линия – Телеком, 2010. – 232 с.

26. *Шеннон К.* Работы по теории информации и кибернетике. – М.: Иностранная литература, 1963. – 832 с.

27. Courtois N., Pieprzyk J. Cryptanalysis of Block Ciphers with Overdefined Systems of Equations // Proc. Asiacrypt-02. – Lect. Notes in Comp. Sci. – 2002. – Vol. 2501. – P. 267 – 287.

28. *Зензин О.С.* Стандарт криптографической защиты – AES. Конечные поля. / О.С. Зензин, М.А. Иванов – М.: КУДИЦ-ОБРАЗ, 2002. – 176 с.

29. *Matsui M.* Linear cryptanalysis method for DES cipher // Advances in Cryptology — EUROCRYPT ’93, LNCS, v. 765, 1994, pp. 386–397.

30. *Sorkin A.* Lucifer, a Cryptographic Algorithm // Cryptologia. - 1984. - Vol. 8, № 1. - P. 22-41.

31. Nyberg K., Knudsen L. Provable Security Against a Differential Attack // J. of Cryptology. - 1995. - Vol. 8. - P. 27-37.

32. *Олейников Р.В.* Дифференциальный криптоанализ алгоритма шифрования ГОСТ 28147-89 // Радиотехника, 2001 (119). – С. 146-152.

33. *Schneier B.* Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish) // Proc. Fast Software Encryption-93. - Lect. Notes in Comp. Sci. - 1994. - Vol. 809. - P. 191-204.

34. Burwick C. et al. MARS - a candidate cipher of AES // AES submission. - 1999.

35. *Adams C.* The CAST-256 Encryption Algorithm // AES submission. - 1998.

36. Khovratovich D., Nikolic I. Rotational Cryptanalysis of ARX // Proc. Fast Software Encryption-10. - Lect. Notes in Comp. Sci. - 2010. - Vol. 6147. - P. 333-346.

37. *Аграновский А.В.* Практическая криптография: алгоритмы и их программирование / А.В. Аграновский, Р.А. Хади – М.: СОЛОН_Пресс, 2002. – 256 с.

38. ГОСТ 28147-89. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования. – М.: Изд-во стандартов, 1989. – 20 с.

39. *Черемушкин А.В.* Криптографические протоколы. Основные свойства и уязвимости. - М.: Изд. дом "Академия", 2009. - 272 с.

40. National Institute of Standards and Technology (NIST). FIPS Publication 197: Advanced Encryption Standard (AES). Nov. 2001.

41. Biham E., Shamir A. Differential cryptanalysis of DES-like cryptosystems of DES-like cryptosystems // Advances in Cryptology _ CRYPTO ’90. Lecture Notes in Computer Science. Springer–Verlag. 1991. Vol. 537. P. 2–21.

42. *Biham E.* Enhancing differential-linear cryptanalysis / E. Biham, O. Dunkelman, and N. Keller, – NESSIE, 2002. NES/DOC/TEC/WP5/017.

43. *Диффи У.* Новые направления в криптографии. / У. Диффи, М.Э. Хеллмэн; // ТИИЭР. 1976.

44. *Торстейнсон П.* Криптография и безопасность в технологии .NET: Пер. с англ. - М.: БИНОМ. 2007 - 479 с.

45. *Горбенко И.Д.* Критерии отбора случайных таблиц подстановок для алгоритма шифрования по ГОСТ 28147-89 / И.Д. Горбенко, И.В. Лисицкая //. Радиотехника. Всеукр. межвед. науч-техн. сб. 1997. Вып. 103. С. 121–130.

46. ГОСТ Р 34.12-2015. Информационная технология. Криптографическая защита информации. Блочные шифры. М.: Стандартинформ, 2015.

47. СТБ 34.101.31–2011. Информационные технологии и безопасность. Защита информации. Криптографические алгоритмы шифрования и контроля целостности. – Взамен СТБ П 34.101.31–2007; введ. 31–01–2011. – Минск, 2011. – 35 с.

48. *Венбо Мао.* Современная криптография: теория и практика. / Венбо Мао. – М.: Изд. дом «Вильямс», 2005. – 768 с.

49. *Исагулиев К.П.* Справочник по криптологии. / Исагулиев К.П. – Мн.: Новое знание, 2005. – 237 с.

50. *Горбенко І.Д.* Криптографічний захист інформації. / Горбенко І.Д.; Навч. посібник. – Харків: ХНУРЕ, 2004 р.

51. *Молдовян А.А.* Криптография: скоростные шифры. / Молдовян А.А., Молдовян Н.А., Гуц Н.Д., Изотов Б.В. – СПБ.: БХВ-Петербург, 2002. – 496 с.

52. *Есин И.И.* Безопасность информационных систем и технологий / Есин И.И., Кузнецов А.А., Сорока Л.С. – Х.: ЭДЕНА, 2010. – 656 с.

53. *Goldreich O.* Foundations of cryptography. Vol.1. Basic tools. Cambridge University Press. 2005.

54. *Шнайер Б.* Секреты и ложь. Безопасность данных в цифровом мире. / Б. Шнайер. – СПб.: Петер, 2003. – 3688 с.

55. *Масленников М.Е.* Практическая криптография. / М.Е. Масленников – СПБ.: БХВ-Петербург, 2005. – 464 с.

56. Kim J., Hong S., Preneel B. Related-key rectangle attacks on reduced in Comp. Sci. - 2007. - Vol. 4593. - P. 225-241.

57. Biham E., Shamir A. Differential cryptanalysis of the full 16-round DES // Proc. Crypto-92. - Lect. Notes in Comp. Sci. - 1993. - Vol. 740. - P.487-496.

58. Biham E., Biryukov A., Shamir A. Miss in the Middle Attacks on IDEA and Khufu // Proc. Fast Software Encryption-99. - Lect. Notes in Comp. Sci. - 1999. - vol. 1636. - P. 124-137.

59. Wagner D. The boomerang Attack // Proc. Fast Software Encryption-99. - Lect. Notes in Comp. Sci. - 1999. - Vol. 1636. - P. 156-170.

60. Вильям Столингс. Криптография и защита сетей. Принципы и практика. / Вильям Столингс. – Изд. Дом “Вильямс”.М.С-П.К. – 2001. – 670с.

61. Nybery K. Linear Approximations of Block Ciphers // Lecturt Notes in Computer Science. Springer-Verlag. - 1994. - V. 950. - P. 439-444.

62. Knudsen L., Wagner D. Integral cryptanalysis // Proc. Fast Software Encryption-02. - Lect. Notes in Comp. Sci. - 2002. - vol. 2365. - P. 629-632.

63. Кнут Д. Исскуство программирования. - Т.2. Получисленные алгоритмы: Пер. с англ. - М.: Изд. дом "Вильямс", 2007. - 832 с.

64. Иванов М.А. Теория, применение и оценка качества генераторов псевдослучайных последовательностей. / Иванов М.А., Чугунков И.В. – М.: КУДИЦ-ОБРАЗ, 2003. – 240 с.

65. Горбенко І.Д. Криптографічний захист інформації. / Горбенко І.Д.; Навч. посібник. – Харків: ХНУРЕ, 2004 р.

66. Эвристические алгоритмы и распределённые вычисления в прикладных задачах (Выпуск 2): коллективная монография / Под ред. Б.Ф. Мельникова. – Ульяновск, 2013. – 202 с.

67. Белецкий А.А. Криптографические примитивы, основанные на методе скользящего кодирования. / Белецкий А.Я., Белецкий А.А. // Вісник СумДУ, 2007. – № 10. – С. 29-38.

68. Соколов Н.П. Введение в теорию многомерных матриц. / Н. П. Соколов. - К.: 1972. - 175 с.

69. Соколов Н.П. Пространственные матрицы и их приложения / Н. П. Соколов. - М.: 1960. - 300 с.

70. Соколов А. Новые методы синтеза нелинейных преобразований современных шифров // Одесса. – 2015. – 100 с.

71. Патент України на корисну модель № 94189, МПК G09C 1/00 (2014.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201312117 ; заявл. 16.10.2013; опубл. 10.11.2014, Бюл.№ 21 – 5 с. : іл.

72. Патент України на корисну модель № 95753, МПК G09C 1/00 (2015.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201406124 ; заявл. 04.06.2014; опубл. 12.01.2015, Бюл.№ 1 – 5 с. : іл.

73. Патент України на корисну модель № 98731, МПК G09C 1/00 (2015.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201410960 ; заявл. 07.10.2014; опубл. 12.05.2015, Бюл.№ 9 – 5 с. : іл.

74. Патент України на корисну модель № 99696, МПК G09C 1/00 (2015.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201404060 ; заявл. 16.04.2014; опубл. 25.06.2015, Бюл.№ 12 – 5 с.

75. Патент України на корисну модель № 99698, МПК G09C 1/00 (2015.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201404063 ; заявл. 16.04.2014; опубл. 25.06.2015, Бюл.№ 12 – 5 с.

76. Патент України на корисну модель № 113149, МПК G09C 1/00 (2016.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201608329 ; заявл. 28.07.2016; опубл. 10.01.2017, Бюл.№ 1 – 5 с.

77. Патент України на корисну модель № 113150, МПК G09C 1/00 (2016.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201608330 ; заявл. 28.07.2016; опубл. 10.01.2017, Бюл.№ 1 – 5 с.

78. Патент України на корисну модель № 113464, МПК G09C 1/00 (2016.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201608330 ; заявл. 28.07.2016; опубл. 10.01.2017, Бюл.№ 2 – 5 с.

79. Патент України на корисну модель № 113465, МПК G09C 1/00 (2016.01). Спосіб криптографічного перетворення інформації / Білецький А.Я., Навроцький Д.О.; заявник і патентовласник Національний авіаційний університет. – № u201608330 ; заявл. 28.07.2016; опубл. 10.01.2017, Бюл.№ 1 – 5 с.

80. Rukhin A. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications./ A.Rukhin, J.Soto. – NIST Special Publication 800-22, 09.2000.

81. Robert G. Brown. Dieharder: A Random Number Test Suite / Duke University Physics Department. – 2015.

82. Кінзерявий В.М. Верхні оцінки стійкості блокових шифрів із рандомізованими вузлами заміни до методів лінійного та диференціального криптоаналізу / В.М. Кінзерявий // Захист інформації. - 2013. - т. 15, № 1. - С. 21-31.

83. *Навроцький Д.О.* Методи комп'ютерної стеганографії / Д. О. Навроцький // Вісник Національного технічного університету України «Київський політехнічний інститут» – Серія «Радіотехніка. Радіоапаратобудування». – 2007. – № 35. – С. 105–108.

84. *Навроцький Д.О.* Представлення і прогнозування ефективності нового протоколу оцінки якості реалізації розроблюваних алгоритмів комп'ютерної стеганографії / Д. О. Навроцький // Вісник Національного технічного університету України «Київський політехнічний інститут» – Серія «Радіотехніка. Радіоапаратобудування». – 2007. – № 34. – С. 150–156.

85. Концепція (основи державної політики) національної безпеки України: (Із змінами, внесеними згідно з Постановою КМ N 1849 (1849-98-п) від 23.11.98) (Відомості Верховної Ради (ВВР), 1997, N10, ст.85).

86. Закон України "Про державну таємницю": Відомості Верховної Ради (ВВР), 1994, N 16, ст.93. – (Із змінами, внесеними згідно із Законами N 971-IV від 19.06.2003, ВВР, 2003, N 45, ст.361 N 1519-IV від 19.02.2004). [Електронний ресурс] – Режим доступу: http://www.dstszi.gov.ua/dstszi/control/uk/publish/article?art_id=40966&cat_id=38828

87. Закон України "Про Національну систему конфіденційного зв'язку": (Із змінами, внесеними згідно із Законами N 1280-IV від 18.11.2003, ВВР, 2004, N 12, ст.155 N 2599-IV від 31.05.2005). [Електронний ресурс] – Режим доступу:http://dstszi.gov.ua/dstszi/control/uk/publish/article?art_id=43399&cat_id=38828

88. Положення про порядок здійснення криптографічного захисту інформації в Україні. (Затверджено Указом Президенту України № 505 від 22.05. 1998 року). [Електронний ресурс] – Режим доступу: http://www.dstszi.gov.ua/dstszi/control/uk/publish/article?showHidden=1&art_id=38901&cat_id=38829&ctime=1127720611472

89. Закон України «Про Концепцію Національної програми інформатизації» (Відомості Верховної Ради (ВВР), 1998, N 27-28, ст.182) (Із змінами, внесеними згідно із Законом N 3421-IV (3421-15) від 09.02.2006, ВВР, 2006, N 22, ст.199). [Електронний ресурс] – Режим доступу: http://zakon.rada.gov.ua/cgi-bin/laws/main.cgi?nreg=75%2F98-%E2%F0

90. Белкин Т.Г. Способ скоростного шифрования на базе управляемых операций / Т.Г. Белкин, Н. Д. Гуц, А. А. Молдовян, Н. А. Молдовян // Управляющие системы и машины. 1999. — № 6. — С. 78-87.

91. Волкович С.Л. Вступ до алгебраїчної теорії перешкодостійкого кодування / Волкович С. Л., Геранін В. О., Мовчан Т. В., Пісаренко Л. Д. – Київ, ВПФ УкрІНТЕІ, 2002. – 236 с.

92. Гантмахер Ф.Р. Теория матриц. / Ф.Р. Гантмахер. — 5-е изд. — М.: Физматлит, 2005. — 560 с.

93. Гуц Н.Д. Построение управляемых блоков перестановок с заданными свойствами / Н. Д. Гуц, А. А. Молдовян, Н. А. Молдовян // Вопросы защиты информации. 1999. - № 5. - С. 39-49.

94. Повідомлення організаційного комітету по проведенню відкритого конкурсу криптоалгоритмів про припинення прийому заявок на участь у конкурсі. [Електронний ресурс] – Режим доступу: http://dstszi.gov.ua/dstszi/control/uk/publish/article?art_id=49027&cat_id=38710

95. Асосков А.В. Поточные шифры / Асосков А.В., Иванов М.А., Мирский А.А., Рузин А.В. – М.: КУДИЦ-ОБРАЗ, 2003. – 336 с.

96. Брикелл Э.Ф. Криптоанализ: обзор новейших результатов. / Брикелл Э.Ф., Одлижко Э.М. // ТИИЭР: Малый тематический выпуск. Защита информации. 1988.76(5).

97. Горбенко И.Д. Анализ каналов уязвимости системы RSA / Горбенко И.Д., Долгов В.И., Потий А.В. Федорченко В.Н. // Безопасность информации. – 1995. №2. – С. 22-26.

98. Диффи У. Хеллман М. Защищенность и имитостойкость: Введение в криптографию // ТИИЭР, - т.67, - №3, 1979. - С. 79-109

99. Долгов В.И. Анализ циклических свойств блочных шифров / Долгов В.И., Руженцев И.В., Лисицкая И.В. // Прикладная радиэлектроника, – Харьков, – 2007 г., №2, С. 257-263.

100. Долгов В.И. О некоторых подходах к построению безусловно стойких кодов аутентификации коротких сообщений / Долгов В.И. Федорченко В.Н. // Управление и связь. – Х.: НАНУ, ПАНИ, ХВУ – 1996. – С. 47-51.

101. Исагулиев К.П. Справочник по криптологии. / Исагулиев К.П. – Мн.: Новое знание, 2005. – 237 с.

102. Поточные шифры. Результаты зарубежной открытой криптологии. Электронный ресурс: http//www.ssl.stu.neva.ru/psw/crypto.html.

103. Роджерс Д., Адамс Дж. Математические основы машинной графики: Пер. с англ. – М.: Мир, 2001. – 604 с.

104. Ростовцев А.Г. Теоретическая криптография. / – Санкт-Петебург: «Профессионал», 2005. – 480 с.

105. Ростовцев А.Г. Два подхода к анализу блочных шифров / Ростовцев А.Г., Маховенко Е.Б. // Проблемы информационной безопасности. Компьютерные системы. СПб., 2002, № 1. – С. 49-55.

106. Белецкий А.А. Алгоритмы построения криптографических нелинейных подстановок, управляемых состоянием ключа шифрования. / Белецкий А.Я., Белецкий А.А., Кузнецов А.А. // Електроніка та системи управління, 2007. – № 3(13). – С. 5-9.

107. Ростовцев А.Г. О матричном шифровании (критика протокола Ероша и Скуратова). – [Электронный ресерс] – Режим доступа: www.ssl.stu.neva.ru/psw/crypto/rostovtsev/Erosh_Skuratov.pdf

108. Белецкий А.А. Анализ эффективности симметричной СЗПЗС криптографического алгоритма защиты даннях / Белецкий А.Я., Белецкий А.А. // Електроніка та системи управління, 2005. – № 2(4). – С. 17-25.

109. Белецкий А.А. Симметричный блочный RSB-32 криптоалгоритм. / Белецкий А.Я., Белецкий А.А. // Захист інформації, 2005. – № 2. – С. 42-50.

110. Белецкий А.А. Семейство симметричных блочных RSB криптографических алгоритмов с динамически управляемыми параметрами шифрования. / Белецкий А.Я., Белецкий А.А., Кузнецов А.А. // Електроніка та системи управління, 2007. – № 1(11). – С. 5-16.

111. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. / Б. Шнайер. – М.: ТРИУМФ, 2002. – 816 с.

112. Белецкий А.А. Семейство Уолша генераторов поточного блочно-сбалансированного шифрования. / Белецкий А.Я., Белецкий А.А., Кузнецов А.А. // Захист інформації, 2007. – № 2(33). – С. 42-55.

113. Шниперов А.Н. Синтез и анализ высокоскоростных симметричных криптосистем на основе управляемых операций / А. Н. Шниперов // Информационные технологии. М.: изд-во «Новые технологии», 2008. - № 1. — С. 36-41.

114. Юдін О.К. Захист інформації в мережах передачі даних / Юдін О.К., Корченко О.Г., Конахович Г.Ф. – К.: ИНТЕРСЕРВИС, 2009. – 716 с.

115. Ященко В.В. Основные понятия криптографии / Ященко В.В. // Математическое просвещение, 1998. – Сер. 3, вып. 2. – С. 53-70.

116. Hoffstein J., Pipher J., Silverman J. H. An Introduction to Mathematical Cryptography. Springer. 2008.

117. National Institute of Standards and Technology (NIST). FIPS Publication 46-3: Data Encryption Standard (DES). Jan. 1999.

118. Niederreiter H. Knapsack-type cryptosystems and algebraic coding theory // Problems of Control and Information Theory. – 1986. - P. 159-166.

119. Ohta K. and Koyama K. Meet-in-the-Middle Attack on Digital Signature Schemes. In Abstract of AUSCRYPT ’90, 1990. P. 110-121.

120. Rivest R.L. A method for obtaining digital signatures and public key cryptosystems / Rivest R.L., Shamir A., Adleman I.M. // Comm. ACM. - 1978. - P. 120-126.

121. Rivest R.L. The RC5 Encryption Algorithm / R. L. Rivest. – Fast Software Encryption, Second International Workshop // Lecture Notes in Computer Science. Springer-Verlag. 1995. - Vol. 1008. - P. 86-96.

122. Rivest R. L. The RC6 Block Cipher / R. L. Rivest, M. J. B. Robshaw, R. Sidney, Y. L. Yin – Proceedings of the 1st Advanced Encryption Standard. Candidate Conference, Venture, California, Aug. 20-22, 1998. – [Электронный

ресурс] Режим доступа: http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf

123. Stinson D.R. Cryptography. Theory and practice. 3 ed. Chapman & Hall/CRC. 2006.

124. *Навроцький Д.О.* Шифратор на МК під'єднаний до ПК через USB / О.Д. Навроцький // Сучасні тенденції розвитку системного програмування: науково-практична конференція. – 25-26 листоп. 2015р.: тези доп. – К., 2015. – С. 82.

125. *Навроцький Д.О.* Перехоплення і модифікація даних за допомогою перехідника USB-UART / Д.О. Навроцький // ITSEC: матеріали V міжнародної науково-технічної конференції. – 19-22 трав. 2015р.: тези доп. – К.: НАУ, 2015. – С. 36.

126. *Навроцький Д.О.* Несанкціоноване перехоплення і модифікація даних каналу зв'язку БПЛА / Д. О. Навроцький // Матеріали XII міжнародної науково-технічної конференції «АВІА-2015». – 28–29 квіт. 2015р.: тези доп. – К. НАУ, 2015. – Т.1. – С. 2.50–2.53.

127. *Навроцький Д.О.* Захист радіоканалів БПЛА від несанкціонованого втручання / Д. О. Навроцький // Проблеми розвитку глобальної системи зв'язку, навігації, спостереження та організації повітряного руху CNS/ATM: науково-практична конференція. – 17–19 листоп. 2014 р.: тези доп. – К., 2014. – С.164.

128. *Навроцький Д.О.* Криптоаналіз з застосуванням ланцюгових дробів / Д. О. Навроцький // Розвиток наукових досліджень 2013: IX міжнародна науково-практична конференція. – 25–27 листоп. 2013 р.: тези доп. – К., 2013. – С.9.

129. *Navrotskyi D.O.* Hardware and software complex with encrypted communication channel of Ground-to-UAV-to-Ground / D.O. Navrotskyi //

Проблеми навігації і управління рухом: Всеукраїнська науково-практична конференція молодих учених і студентів. – 18–20 листоп. 2013 р.: тези доп. – К., 2013. – С.9.

130. *Navrotskyi D.O.* Ground-to-UAV-to-Ground hardware and software complex with encrypted communication channel / D.O. Navrotskyi // System analysis and information technologies (SAIT): 15-th International conference SAIT 2013, Kyiv, Ukraine, May 27–31, 2013. Proceedings. – ESC «IASA» NTUU «KPI», 2013. – 516 p.

131. *Навроцький Д.О.* Апаратно-програмний комплекс «Земля – БПЛА – Земля» з захищеним шифрованим каналом зв'язку / Д. О. Навроцький // Наукоємні технології: Науково-технічна конференція студентів та молодих учених, 12–16 листоп. 2012 р.: тези доп. – К., 2012. – С. 9.

132. *Навроцький Д.О.* Апаратно-програмний комплекс із шифрованим каналом зв'язку «Земля – БПЛА – Земля» / Д. О. Навроцький // Проблеми та перспективи розвитку авіації та космонавтики: Всеукраїнська науково-практична конференція молодих учених і студентів, 24–25 жовт. 2012 р.: тези доп. – К., 2012. – С. 223.

133. *Beletsky A. Ja.* Matrix analogues of the Diffi-Hellman protocol / A. Ja. Beletsky, O.I. Volivach, R. Ju. Kandiba, D. O. Navrotskyi // Safety in Aviation and Space Technologies: Proceedings. The firth world congress «Aviation in the XXI-st century», September 25-27, 2012,: abstracts, – K., 2012. – P. 1.7.5 – 1.7.9.

ДОДАТОК 1. Статистичні таблиці RSB-64-3D шифру.

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
11	9	2	15	10	10	13	9	14	7	0.181557	0.9800	frequency
13	8	15	8	3	7	13	13	9	11	0.213309	1.0000	block-frequency
6	10	12	13	11	14	8	10	2	14	0.162606	0.9900	cumulative-sums
11	7	10	12	6	14	12	9	10	9	0.816537	0.9800	cumulative-sums
10	8	11	12	5	13	11	10	10	10	0.883171	1.0000	runs
11	13	12	8	10	11	7	12	8	8	0.911413	1.0000	longest-run
10	4	14	7	7	8	9	13	17	11	0.145326	0.9900	rank
12	14	8	9	7	9	8	10	13	10	0.851383	0.9900	fft
10	9	8	11	10	11	8	15	9	9	0.924076	1.0000	nonperiodic-templates
14	8	4	12	8	11	14	6	10	13	0.304126	1.0000	nonperiodic-templates
15	8	11	12	7	11	10	12	3	11	0.366918	0.9900	nonperiodic-templates
9	16	5	10	14	7	9	10	11	9	0.437274	1.0000	nonperiodic-templates
9	15	10	12	6	8	17	9	4	10	0.137282	0.9900	nonperiodic-templates
8	3	15	12	13	8	9	11	11	10	0.366918	1.0000	nonperiodic-templates
10	16	11	7	6	7	7	13	16	7	0.145326	0.9900	nonperiodic-templates
11	9	11	11	13	10	9	9	9	8	0.991468	0.9900	nonperiodic-templates
8	11	14	17	8	7	9	7	11	8	0.366918	1.0000	nonperiodic-templates
12	6	9	12	11	7	13	8	11	11	0.834308	1.0000	nonperiodic-templates
12	10	12	11	11	6	13	8	8	9	0.883171	0.9800	nonperiodic-templates
5	8	14	11	14	11	3	16	7	11	0.071177	1.0000	nonperiodic-templates
6	10	10	8	7	12	7	23	10	7	0.008879	0.9900	nonperiodic-templates
4	11	13	15	4	8	9	16	10	10	0.096578	1.0000	nonperiodic-templates
11	15	13	6	6	15	8	7	10	9	0.304126	0.9700	nonperiodic-templates
8	15	9	11	8	11	6	13	9	10	0.719747	0.9700	nonperiodic-templates
6	11	11	7	9	18	10	9	10	9	0.401199	1.0000	nonperiodic-templates
11	7	11	11	14	7	9	9	13	8	0.816537	0.9700	nonperiodic-templates
3	8	10	9	13	12	5	12	10	18	0.066882	1.0000	nonperiodic-templates
9	5	5	12	13	6	14	16	10	10	0.153763	1.0000	nonperiodic-templates
10	14	8	15	11	7	11	9	7	8	0.637119	0.9900	nonperiodic-templates
9	11	12	11	6	15	9	8	8	11	0.759756	0.9800	nonperiodic-templates
11	11	11	10	9	7	9	6	8	18	0.366918	0.9900	nonperiodic-templates
11	13	8	10	9	6	5	14	17	7	0.162606	1.0000	nonperiodic-templates
11	9	12	10	7	12	10	11	10	8	0.983453	1.0000	nonperiodic-templates
15	6	9	9	12	12	9	9	9	10	0.798139	1.0000	nonperiodic-templates
14	11	3	10	10	7	8	9	17	11	0.162606	0.9900	nonperiodic-templates
14	10	10	9	12	5	8	12	11	9	0.779188	0.9900	nonperiodic-templates
9	7	11	9	10	9	11	11	13	10	0.983453	1.0000	nonperiodic-templates
10	11	8	17	11	4	7	12	11	9	0.304126	1.0000	nonperiodic-templates
11	8	10	8	13	10	10	5	14	11	0.739918	0.9800	nonperiodic-templates
7	9	8	13	9	4	14	11	12	13	0.437274	0.9700	nonperiodic-templates
7	8	14	13	9	17	7	5	12	8	0.162606	0.9800	nonperiodic-templates
13	12	14	11	7	8	5	15	5	10	0.224821	0.9800	nonperiodic-templates
9	6	17	9	10	5	11	10	10	13	0.334538	0.9700	nonperiodic-templates
9	14	14	4	8	12	14	13	7	5	0.137282	1.0000	nonperiodic-templates
11	8	8	11	14	8	13	9	11	7	0.834308	0.9800	nonperiodic-templates
6	13	5	9	12	18	13	5	13	6	0.037566	0.9900	nonperiodic-templates
13	10	14	15	6	7	6	12	5	12	0.191687	1.0000	nonperiodic-templates
9	11	13	11	8	6	9	11	10	12	0.924076	0.9900	nonperiodic-templates
12	9	13	9	12	8	8	3	10	16	0.262249	0.9700	nonperiodic-templates
5	15	12	8	12	9	12	8	10	9	0.616305	1.0000	nonperiodic-templates
12	11	9	12	12	7	8	14	9	6	0.739918	1.0000	nonperiodic-templates
9	10	6	11	11	15	7	7	9	15	0.455937	0.9900	nonperiodic-templates
6	8	11	11	11	10	17	11	7	8	0.474986	1.0000	nonperiodic-templates
9	13	8	7	6	13	14	10	6	14	0.383827	0.9900	nonperiodic-templates
4	11	8	14	11	7	14	13	8	10	0.383827	1.0000	nonperiodic-templates
6	13	13	6	10	14	11	7	13	7	0.401199	1.0000	nonperiodic-templates
8	14	13	16	7	8	7	11	9	7	0.366918	0.9900	nonperiodic-templates
12	9	4	12	11	16	11	9	9	7	0.401199	0.9900	nonperiodic-templates
10	4	10	14	11	9	14	10	12	6	0.437274	0.9800	nonperiodic-templates

8	9	6	17	15	9	4	11	12	9	0.129620	1.0000	nonperiodic-templates
9	5	11	12	13	10	15	9	9	7	0.574903	0.9900	nonperiodic-templates
10	8	9	11	11	12	10	7	18	4	0.213309	0.9900	nonperiodic-templates
11	12	11	11	12	10	8	5	9	11	0.897763	0.9900	nonperiodic-templates
14	9	9	13	9	8	10	8	8	12	0.883171	1.0000	nonperiodic-templates
13	8	16	8	14	10	7	7	9	8	0.419021	0.9900	nonperiodic-templates
11	8	11	7	9	11	14	8	10	11	0.924076	0.9900	nonperiodic-templates
11	8	10	9	10	8	9	12	13	10	0.983453	0.9900	nonperiodic-templates
11	9	9	10	14	8	10	12	8	9	0.955835	0.9800	nonperiodic-templates
9	13	11	8	10	8	10	9	10	12	0.983453	1.0000	nonperiodic-templates
10	15	6	9	13	3	12	11	9	12	0.275709	1.0000	nonperiodic-templates
8	13	11	10	7	10	8	10	12	11	0.955835	0.9800	nonperiodic-templates
12	12	9	14	14	6	6	12	7	8	0.437274	0.9900	nonperiodic-templates
9	5	11	13	8	10	10	8	13	13	0.719747	0.9900	nonperiodic-templates
9	13	6	8	12	10	12	8	12	10	0.867692	0.9900	nonperiodic-templates
17	8	11	11	10	11	8	5	11	8	0.437274	0.9800	nonperiodic-templates
11	7	8	11	8	8	8	12	16	11	0.657933	1.0000	nonperiodic-templates
11	11	6	7	9	9	12	10	12	13	0.867692	0.9700	nonperiodic-templates
8	9	8	8	13	8	10	13	12	11	0.911413	1.0000	nonperiodic-templates
13	5	7	16	13	13	12	10	7	4	0.102526	0.9900	nonperiodic-templates
10	10	9	11	14	8	12	6	12	8	0.834308	0.9900	nonperiodic-templates
11	12	13	9	10	9	11	4	14	7	0.554420	0.9900	nonperiodic-templates
11	11	12	7	5	16	9	13	9	7	0.383827	0.9800	nonperiodic-templates
10	9	8	11	10	11	8	15	9	9	0.924076	1.0000	nonperiodic-templates
4	13	11	10	13	11	4	14	13	7	0.181557	0.9900	nonperiodic-templates
13	9	5	4	13	9	12	13	9	13	0.319084	0.9800	nonperiodic-templates
6	9	8	10	7	8	14	15	6	17	0.122325	0.9900	nonperiodic-templates
12	7	12	7	15	11	7	9	12	8	0.637119	0.9900	nonperiodic-templates
6	12	12	8	11	15	13	8	8	7	0.534146	1.0000	nonperiodic-templates
7	12	11	12	10	7	7	17	8	9	0.437274	1.0000	nonperiodic-templates
11	11	21	7	6	8	5	16	7	8	0.007160	0.9800	nonperiodic-templates
16	10	17	5	10	8	7	10	7	10	0.153763	0.9800	nonperiodic-templates
7	10	9	14	10	9	8	12	11	10	0.935716	0.9900	nonperiodic-templates
8	11	13	9	13	7	9	9	9	12	0.911413	0.9900	nonperiodic-templates
13	2	8	12	7	5	9	12	14	18	0.017912	1.0000	nonperiodic-templates
9	9	9	12	12	11	14	12	6	6	0.699313	1.0000	nonperiodic-templates
4	8	14	15	11	7	8	15	8	10	0.191687	1.0000	nonperiodic-templates
8	12	10	11	4	11	11	10	9	14	0.699313	1.0000	nonperiodic-templates
10	13	14	5	11	10	8	11	9	9	0.759756	0.9800	nonperiodic-templates
12	12	13	4	7	12	9	11	11	9	0.637119	0.9800	nonperiodic-templates
12	11	11	9	9	10	7	8	13	10	0.964295	0.9600	nonperiodic-templates
14	11	10	9	10	11	6	12	8	9	0.883171	0.9800	nonperiodic-templates
15	12	9	6	15	7	8	9	7	12	0.366918	1.0000	nonperiodic-templates
12	11	8	10	12	10	11	11	6	9	0.955835	0.9700	nonperiodic-templates
15	10	14	6	9	10	9	14	6	7	0.350485	0.9900	nonperiodic-templates
9	9	18	8	8	8	10	8	13	9	0.419021	1.0000	nonperiodic-templates
11	19	10	5	10	10	7	7	9	12	0.162606	0.9700	nonperiodic-templates
12	8	5	14	7	13	10	7	13	11	0.474986	0.9900	nonperiodic-templates
9	10	12	9	9	9	12	5	13	12	0.834308	1.0000	nonperiodic-templates
14	12	13	8	10	8	10	8	6	11	0.759756	1.0000	nonperiodic-templates
12	14	9	11	11	6	8	4	13	12	0.419021	0.9900	nonperiodic-templates
10	9	6	11	9	6	12	12	15	10	0.657933	1.0000	nonperiodic-templates
8	16	14	8	10	10	10	5	11	8	0.437274	1.0000	nonperiodic-templates
7	8	11	7	16	7	17	8	14	5	0.062821	0.9800	nonperiodic-templates
14	8	8	11	8	6	14	9	13	9	0.616305	0.9800	nonperiodic-templates
8	9	8	8	14	12	12	8	11	10	0.897763	0.9900	nonperiodic-templates
10	12	11	10	7	9	12	12	12	5	0.816537	0.9900	nonperiodic-templates
10	7	10	8	10	10	13	9	11	12	0.971699	0.9800	nonperiodic-templates
10	11	6	7	10	10	8	13	12	13	0.816537	0.9800	nonperiodic-templates
7	7	13	10	10	7	5	13	14	14	0.334538	0.9900	nonperiodic-templates
8	12	11	6	10	9	7	13	12	12	0.816537	0.9900	nonperiodic-templates
7	14	12	19	9	8	8	10	8	5	0.096578	0.9900	nonperiodic-templates
14	7	8	15	8	8	11	10	7	12	0.574903	0.9800	nonperiodic-templates
17	11	8	8	15	8	14	5	7	7	0.102526	0.9800	nonperiodic-templates
8	10	6	12	11	13	15	10	5	10	0.494392	1.0000	nonperiodic-templates
14	11	12	9	8	7	8	12	11	8	0.851383	0.9900	nonperiodic-templates
12	15	11	5	7	7	9	8	11	15	0.319084	0.9700	nonperiodic-templates
8	9	9	12	14	13	6	5	7	17	0.145326	0.9800	nonperiodic-templates

```
  9  11  10   8   9  13  12  12   6  10  0.911413   0.9800    nonperiodic-templates
 10   9   7   6  16   8  10  13  10  11  0.574903   1.0000    nonperiodic-templates
 12   8   7  12  15   9   7   7  13  10  0.595549   0.9900    nonperiodic-templates
 10  12   5   8  15   9  10  10  14   7  0.494392   0.9700    nonperiodic-templates
 13  11   5  13  11   9   6   8   9  15  0.419021   1.0000    nonperiodic-templates
  8  15   9   7  10   6  13   9  14   9  0.514124   0.9900    nonperiodic-templates
  8  13   8  15  11   6   8   9  13   9  0.595549   1.0000    nonperiodic-templates
  8  10  10  10   8   9  13  10   9  13  0.971699   1.0000    nonperiodic-templates
 10  13  11   9   8   8   5  16  11   9  0.514124   1.0000    nonperiodic-templates
 10  12   9  10   7  12  14   9  10   7  0.883171   0.9900    nonperiodic-templates
 12  14  10  11   8   7  12   7  10   9  0.851383   1.0000    nonperiodic-templates
  8  14  13   6  11  14   5   7  10  12  0.350485   1.0000    nonperiodic-templates
  8  12  10   9  12   7   9  13  10  10  0.955835   1.0000    nonperiodic-templates
 13   6  12   6   9  14   6  11  17   6  0.108791   1.0000    nonperiodic-templates
 12   9   6   8  10  13  13  11  11   7  0.798139   0.9900    nonperiodic-templates
  7   8  15  10  11  11   8  10  10  10  0.883171   1.0000    nonperiodic-templates
  5  15  11   8  10   7  11  11  10  12  0.637119   0.9900    nonperiodic-templates
 10   8   5   8   7  13  19  11  13   6  0.071177   1.0000    nonperiodic-templates
 12   7   6  12   9  12  10   9  10  13  0.851383   0.9900    nonperiodic-templates
  9   8  16  11   8  14  12   7   5  10  0.350485   0.9800    nonperiodic-templates
  4  15  13  14  11   6   4   9  11  13  0.090936   0.9900    nonperiodic-templates
 11  12  15   8  10   4   8  11   6  15  0.236810   0.9800    nonperiodic-templates
 19   9  13   9   8  11  11   1   9  10  0.035174   0.9800    nonperiodic-templates
  5  12  12  15   7   9   8  15   8   9  0.334538   1.0000    nonperiodic-templates
  7  13  16  11   9   6  11   8  11   8  0.514124   0.9800    nonperiodic-templates
 15   9   9   3  10  13   3  16   7  15  0.015598   0.9900    nonperiodic-templates
  8  12   8  14  13   9   9   8   8  11  0.851383   0.9900    nonperiodic-templates
 11   7  13   6   9  10  11  10  15   8  0.678686   0.9700    nonperiodic-templates
 11  11  12   7   5  16   9  13  10   6  0.334538   0.9800    nonperiodic-templates
 15  15   9   9   8  11  11   9  10   3  0.289667   0.9900    overlapping-templates
  4  18   9   9  14  12   7  12   7   8  0.096578   1.0000    universal
  4  13   6  13  12   9  13   9  11  10  0.474986   0.9900    apen
  9   4   7   2   5   7   9   8   8   7  0.500934   0.9848    random-excursions
  6   6   5   8   8   5   5   3  10  10  0.468595   0.9697    random-excursions
  7   5   4   9   3   8  10   6   6   8  0.534146   0.9848    random-excursions
  8   7   6   6   4  12   6   5   7   5  0.534146   1.0000    random-excursions
  7   4   8   8   8   6   4   6   9   6  0.834308   1.0000    random-excursions
  3   9   4   7   8   9   8   8   6   4  0.534146   1.0000    random-excursions
  7   4   9   7   6   5   7   9   5   7  0.862344   1.0000    random-excursions
 10   5   7   3   8  11   6   9   4   3  0.162606   1.0000    random-excursions
  6   8  12   8   5   3   8   6   4   6  0.324180   0.9848    random-excurs-variant
  7  10   7  11   6   3   1   4   9   8  0.074177   0.9697    random-excurs-variant
  7  10   7   8   4   7   3   6   8   6  0.671779   0.9697    random-excurs-variant
  8   2   7  13   7   7   9   2   6   5  0.060239   0.9697    random-excurs-variant
  4   6   3   9   7  14   9   2   4   8  0.017912   0.9697    random-excurs-variant
  5   5   7   4   8   5  10  11   6   5  0.437274   0.9697    random-excurs-variant
  4   6  10   5   5   5   7   8   8   8  0.739918   1.0000    random-excurs-variant
  5   4   9   5   9   1   4   7  13   9  0.028181   1.0000    random-excurs-variant
  8   8   6   6   3   8   6   5  10   6  0.706149   1.0000    random-excurs-variant
  9   6   9   5   8   3   8   5   8   5  0.637119   1.0000    random-excurs-variant
  7   6   5  10   7  11   4   7   2   7  0.275709   1.0000    random-excurs-variant
  6   7   5   5   5   9   4   7   8  10  0.706149   1.0000    random-excurs-variant
  6   4   5   7   9   3   6  10   9   7  0.500934   1.0000    random-excurs-variant
  6   7   8   4   3   9   6   6   6  11  0.468595   1.0000    random-excursons-variant
  9   5   8   4   9   2   6   9   9   5  0.324180   1.0000    random-excurs-variant
  7   4   7   6  10   6   8   3   6   9  0.602458   1.0000    random-excurs-variant
  6   4   9   8   4   6   7   9   6   7  0.804337   1.0000    random-excurs-variant
  6   6  10   6   5   9   6   9   5   4  0.671779   1.0000    random-excurs-variant
  7  11   7   9   9   9  12  10  17   9  0.574903   1.0000    serial
  7   8  13   7  10  11  10   6  13  15  0.514124   1.0000    serial
 12   2  13   9  14   5   9  11  16   9  0.071177   1.0000    linear-complexity
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the random
excursion (variant) test is approximately = 0.960150 for a sample size = 100
binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately
0.953258 for a sample size = 66 binary sequences.
```

ДОДАТОК 2. Статистичні таблиці M3DCrypt шифру.

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
5	5	8	7	11	10	11	12	15	16	0.162606	1.0000	frequency
6	7	11	5	4	11	12	9	12	23	0.001628	1.0000	block-frequency
4	10	4	6	9	9	14	13	15	16	0.040108	1.0000	cumulative-sums
6	5	10	8	7	7	9	14	17	17	0.037566	1.0000	cumulative-sums
10	10	14	12	15	7	7	8	4	13	0.262249	1.0000	runs
10	11	11	15	16	8	8	9	6	6	0.319084	1.0000	longest-run
13	11	8	9	11	9	9	7	9	14	0.883171	0.9800	rank
13	11	6	6	12	15	8	10	11	8	0.534146	0.9900	fft
6	9	8	16	9	8	8	9	16	11	0.319084	1.0000	nonperiodic-templates
9	14	12	9	12	9	10	10	8	7	0.911413	0.9900	nonperiodic-templates
12	8	6	14	11	12	8	13	8	8	0.678686	0.9600	nonperiodic-templates
11	8	9	13	11	7	9	11	12	9	0.955835	0.9900	nonperiodic-templates
5	10	17	9	10	6	11	9	9	14	0.275709	0.9900	nonperiodic-templates
16	5	12	10	13	9	10	11	6	8	0.383827	0.9700	nonperiodic-templates
8	9	7	8	8	9	13	13	11	14	0.759756	1.0000	nonperiodic-templates
10	14	11	8	11	12	11	9	9	5	0.798139	0.9800	nonperiodic-templates
11	8	6	9	11	15	14	7	9	10	0.595549	0.9900	nonperiodic-templates
8	11	9	12	17	11	10	9	8	5	0.437274	0.9900	nonperiodic-templates
11	10	10	11	12	11	6	11	11	7	0.946308	0.9700	nonperiodic-templates
16	12	9	13	7	10	11	10	5	7	0.401199	0.9900	nonperiodic-templates
13	10	9	13	11	4	14	9	8	9	0.554420	0.9800	nonperiodic-templates
9	3	11	9	11	16	18	8	11	4	0.021999	1.0000	nonperiodic-templates
10	6	10	10	15	12	5	13	9	10	0.534146	0.9800	nonperiodic-templates
8	7	11	11	13	9	6	8	10	17	0.401199	0.9900	nonperiodic-templates
7	5	4	14	9	17	18	12	12	2	0.001296	0.9800	nonperiodic-templates
12	10	7	13	14	10	5	8	11	10	0.657933	0.9900	nonperiodic-templates
10	11	8	9	15	11	12	7	10	7	0.798139	0.9700	nonperiodic-templates
9	15	8	12	12	9	11	6	11	7	0.678686	1.0000	nonperiodic-templates
10	9	10	8	12	11	10	13	4	13	0.699313	0.9800	nonperiodic-templates
8	10	13	13	4	12	15	7	10	8	0.350485	0.9900	nonperiodic-templates
9	13	11	14	7	7	8	14	8	9	0.637119	0.9900	nonperiodic-templates
6	11	9	19	11	11	7	8	8	10	0.224821	1.0000	nonperiodic-templates
9	8	9	9	12	9	10	12	14	8	0.935716	1.0000	nonperiodic-templates
12	10	8	4	10	8	13	16	9	10	0.401199	0.9900	nonperiodic-templates
10	15	9	7	14	10	11	11	6	7	0.554420	0.9900	nonperiodic-templates
5	16	6	11	10	9	11	10	9	13	0.437274	0.9900	nonperiodic-templates
11	15	8	10	12	7	7	10	12	8	0.739918	0.9800	nonperiodic-templates
10	13	9	5	8	15	11	15	3	11	0.122325	0.9900	nonperiodic-templates
6	17	10	15	7	9	11	8	8	9	0.275709	1.0000	nonperiodic-templates
6	11	12	9	7	7	18	8	9	13	0.224821	0.9900	nonperiodic-templates
7	15	10	8	12	8	13	10	8	9	0.739918	1.0000	nonperiodic-templates
14	14	13	11	4	7	11	9	9	8	0.401199	0.9900	nonperiodic-templates
9	15	8	8	8	10	10	14	10	8	0.759756	0.9900	nonperiodic-templates
10	9	12	8	10	17	9	6	10	9	0.574903	1.0000	nonperiodic-templates
9	15	6	12	12	9	8	8	11	10	0.739918	1.0000	nonperiodic-templates
8	7	14	10	6	13	6	9	15	12	0.350485	0.9900	nonperiodic-templates
9	12	8	8	13	9	9	12	11	9	0.964295	0.9900	nonperiodic-templates
8	9	11	14	10	10	12	13	8	5	0.699313	1.0000	nonperiodic-templates
11	12	5	14	8	10	10	8	11	11	0.779188	1.0000	nonperiodic-templates
5	7	10	15	10	15	5	14	10	9	0.181557	1.0000	nonperiodic-templates
17	6	7	12	13	6	11	9	11	8	0.275709	0.9800	nonperiodic-templates
7	11	16	12	12	8	10	5	13	6	0.289667	1.0000	nonperiodic-templates
15	10	9	9	6	12	8	9	12	10	0.779188	0.9900	nonperiodic-templates
10	5	7	10	13	14	12	11	6	12	0.494392	0.9800	nonperiodic-templates
7	14	7	7	10	6	15	12	13	9	0.366918	1.0000	nonperiodic-templates
3	10	19	11	9	5	11	10	11	11	0.066882	0.9900	nonperiodic-templates
13	12	6	14	10	8	7	12	11	7	0.616305	0.9700	nonperiodic-templates
11	13	4	12	14	11	5	14	6	10	0.191687	0.9900	nonperiodic-templates
20	7	11	7	5	8	13	10	10	9	0.071177	1.0000	nonperiodic-templates

7	13	7	9	12	7	10	16	12	7	0.437274	1.0000	nonperiodic-templates
9	11	13	10	10	9	8	11	9	10	0.994250	0.9700	nonperiodic-templates
9	9	12	16	10	7	12	10	7	8	0.657933	0.9800	nonperiodic-templates
10	9	6	10	12	12	14	13	8	6	0.637119	1.0000	nonperiodic-templates
7	16	8	8	14	13	6	14	6	8	0.162606	0.9800	nonperiodic-templates
11	8	11	16	8	14	13	3	9	7	0.162606	0.9800	nonperiodic-templates
10	9	10	10	13	14	8	10	7	9	0.911413	1.0000	nonperiodic-templates
16	11	6	6	13	9	13	7	11	8	0.334538	0.9800	nonperiodic-templates
16	14	6	7	7	11	10	12	9	8	0.383827	1.0000	nonperiodic-templates
13	7	13	12	7	17	7	4	15	5	0.030806	0.9900	nonperiodic-templates
14	4	11	8	11	13	14	13	7	5	0.181557	0.9900	nonperiodic-templates
10	6	7	8	8	14	12	8	12	15	0.474986	0.9900	nonperiodic-templates
12	8	11	10	11	13	10	5	6	14	0.574903	0.9800	nonperiodic-templates
10	9	7	11	13	8	13	9	8	12	0.897763	1.0000	nonperiodic-templates
8	11	12	11	9	9	9	8	12	11	0.987896	1.0000	nonperiodic-templates
12	6	11	10	14	9	8	14	10	6	0.595549	0.9900	nonperiodic-templates
12	9	8	10	6	9	11	15	15	5	0.334538	1.0000	nonperiodic-templates
13	12	12	6	10	11	7	8	9	12	0.816537	0.9700	nonperiodic-templates
10	9	9	4	12	13	10	7	14	12	0.534146	1.0000	nonperiodic-templates
11	7	10	17	11	7	8	9	7	13	0.419021	1.0000	nonperiodic-templates
7	12	11	8	11	6	13	11	9	12	0.834308	0.9900	nonperiodic-templates
17	9	12	8	10	6	7	10	12	9	0.455937	0.9700	nonperiodic-templates
7	14	13	8	12	9	5	11	6	15	0.275709	1.0000	nonperiodic-templates
6	9	8	16	9	8	8	9	16	11	0.319084	1.0000	nonperiodic-templates
10	10	8	8	11	12	15	14	6	6	0.474986	0.9900	nonperiodic-templates
12	5	11	6	12	12	12	10	13	7	0.574903	1.0000	nonperiodic-templates
11	12	9	9	13	7	13	5	14	7	0.494392	0.9900	nonperiodic-templates
9	9	12	11	15	8	9	9	9	9	0.911413	1.0000	nonperiodic-templates
11	12	8	14	13	5	9	9	12	7	0.595549	0.9900	nonperiodic-templates
13	11	8	7	8	11	9	10	9	14	0.867692	0.9800	nonperiodic-templates
12	15	7	6	14	11	9	10	9	7	0.514124	0.9800	nonperiodic-templates
13	13	9	5	10	12	10	7	10	11	0.759756	0.9900	nonperiodic-templates
9	12	9	10	4	9	11	12	10	14	0.699313	0.9900	nonperiodic-templates
9	9	11	9	10	12	10	10	10	10	0.999777	1.0000	nonperiodic-templates
13	13	11	6	6	13	13	8	13	4	0.224821	0.9800	nonperiodic-templates
8	11	9	7	12	11	11	7	9	15	0.779188	1.0000	nonperiodic-templates
12	12	9	6	12	9	11	10	8	11	0.935716	1.0000	nonperiodic-templates
10	9	15	6	11	10	6	9	12	12	0.657933	0.9800	nonperiodic-templates
10	11	12	11	6	9	13	11	9	8	0.924076	1.0000	nonperiodic-templates
11	11	8	11	8	8	6	14	12	11	0.816537	1.0000	nonperiodic-templates
10	9	11	7	15	7	12	1	15	13	0.058984	0.9900	nonperiodic-templates
11	9	9	14	12	12	13	6	9	5	0.554420	1.0000	nonperiodic-templates
4	14	10	6	6	11	14	12	12	11	0.275709	1.0000	nonperiodic-templates
8	5	16	7	14	11	5	13	11	10	0.181557	0.9900	nonperiodic-templates
7	10	11	9	13	14	11	9	9	7	0.851383	0.9900	nonperiodic-templates
6	9	10	13	7	11	12	15	10	7	0.595549	1.0000	nonperiodic-templates
9	9	6	6	9	9	14	14	13	11	0.554420	1.0000	nonperiodic-templates
9	6	10	6	14	6	12	11	8	18	0.129620	0.9800	nonperiodic-templates
8	11	12	12	13	7	7	13	8	9	0.798139	0.9800	nonperiodic-templates
13	7	13	8	8	6	9	13	15	8	0.437274	0.9900	nonperiodic-templates
6	10	11	10	14	8	10	9	7	15	0.616305	0.9900	nonperiodic-templates
9	10	10	7	14	5	9	9	20	7	0.062821	1.0000	nonperiodic-templates
10	13	8	12	13	7	15	6	9	7	0.474986	1.0000	nonperiodic-templates
7	8	13	14	13	7	13	6	12	7	0.401199	0.9900	nonperiodic-templates
12	15	13	7	9	4	10	11	7	12	0.366918	0.9800	nonperiodic-templates
8	11	11	7	6	10	9	17	13	8	0.401199	0.9900	nonperiodic-templates
8	5	12	10	10	9	15	10	10	11	0.739918	0.9900	nonperiodic-templates
6	9	10	10	14	11	8	9	9	14	0.779188	0.9900	nonperiodic-templates
8	10	12	13	5	6	14	8	10	14	0.401199	0.9800	nonperiodic-templates
13	14	5	10	12	8	9	9	7	13	0.554420	0.9900	nonperiodic-templates
12	11	8	12	7	7	9	13	12	9	0.867692	0.9900	nonperiodic-templates
8	11	10	12	9	11	12	8	7	12	0.955835	0.9800	nonperiodic-templates
14	8	10	10	13	4	6	17	10	8	0.145326	0.9800	nonperiodic-templates
7	12	6	8	13	11	13	10	11	9	0.798139	1.0000	nonperiodic-templates
6	7	10	8	13	11	18	9	5	13	0.129620	1.0000	nonperiodic-templates
15	12	9	11	9	5	6	10	16	7	0.224821	0.9900	nonperiodic-templates
7	13	6	10	16	9	9	12	8	10	0.534146	0.9800	nonperiodic-templates
11	9	8	13	7	8	14	9	10	11	0.867692	1.0000	nonperiodic-templates

```
 11   6   6   9  13  12   9  15   4  15  0.145326   0.9800    nonperiodic-templates
 17  11  10  10   9   6   9   4  13  11  0.249284   0.9600    nonperiodic-templates
 14  16   6  10   8   8   9   7   8  14  0.304126   0.9900    nonperiodic-templates
 10  11  10   5  17   5  16   7   6  13  0.048716   0.9900    nonperiodic-templates
  8  12  13  14  11   6   4  12   9  11  0.419021   0.9900    nonperiodic-templates
 10  13  12  11   9   6   5   9  15  10  0.514124   0.9900    nonperiodic-templates
  6   9  10  14  11   9   8   9   9  15  0.678686   1.0000    nonperiodic-templates
  9   4  11  14   9   6  18  14  10   5  0.040108   1.0000    nonperiodic-templates
 12  15   7  10   6   7  12   9  13   9  0.554420   0.9900    nonperiodic-templates
 13  10  12   7   8   8  16   9   7  10  0.574903   0.9600    nonperiodic-templates
  4   9  16  12  10   7   8  12  16   6  0.102526   1.0000    nonperiodic-templates
  9  13   6   6  15  10  10  12  13   6  0.383827   0.9900    nonperiodic-templates
 11  14  10  10  13   8   8  11   7   8  0.851383   0.9900    nonperiodic-templates
 11  12   9   7  13   8  11   9  11   9  0.955835   0.9700    nonperiodic-templates
 14   4  12  11  15   7  12   6   8  11  0.236810   0.9700    nonperiodic-templates
 12   7  12   7  10  11  13  10  14   4  0.455937   0.9800    nonperiodic-templates
  5  10   8  17  11  15  10   8   9   7  0.224821   1.0000    nonperiodic-templates
  6  15   7  14   5  17   6   7  14   9  0.032923   1.0000    nonperiodic-templates
 10  14   4   9   8  14  13  13  10   5  0.236810   0.9900    nonperiodic-templates
  7  12  10  12  10  10   9   9   9  12  0.983453   1.0000    nonperiodic-templates
  8  11  13   8  10  16   8  10   9   7  0.657933   0.9900    nonperiodic-templates
 14   7  10   8   8  13   6   9  10  15  0.494392   0.9900    nonperiodic-templates
  8  16  15   8   7   6  11   7  13   9  0.249284   1.0000    nonperiodic-templates
  6   4  12  15   9  10  16   8  13   7  0.122325   1.0000    nonperiodic-templates
 10  11   7   7   7   8  10   8  19  13  0.181557   0.9900    nonperiodic-templates
 11  10  11   9   9   6   7  16  10  11  0.678686   1.0000    nonperiodic-templates
  1   9   8   6  10  13  11  12  15  15  0.055361   1.0000    nonperiodic-templates
 13   7  11   9  11   4  10   8  14  13  0.474986   0.9800    nonperiodic-templates
  7  14  13   8  13   8   5  11   6  15  0.224821   1.0000    nonperiodic-templates
 14  10  13   8  11   9  10   4  11  10  0.657933   1.0000    overlapping-templates
 14  14  11  12   9  11   7  11   3   8  0.334538   1.0000    universal
 12   8   9  13   8   8   9   8  12  13  0.883171   0.9300 *  apen
  2   7   6   5   9   9  12   7  14   6  0.063958   0.9870    random-excursions
 10  10   7   3   8  13   6   6   7   7  0.316916   1.0000    random-excursions
  9   9   6   4  10   7   9   9   6   8  0.808725   0.9870    random-excursions
  6   8   7   2   9  14   9   6   5  11  0.090936   1.0000    random-excursions
  4   3  13  12  10   8   2  13   5   7  0.006196   1.0000    random-excursions
  2  10  10   3   7  13  10   8   7   7  0.090936   0.9870    random-excursions
  9   5   8  11   8   7   8   8   7   6  0.901761   0.9870    random-excursions
  6   9   7   9  10   8  10   3  12   3  0.205375   1.0000    random-excursions
  9   6  11   6  12   6   4   6  10   7  0.386280   0.9870    random-excurs-variant
  8   3  10  11  10   9   5   4  11   6  0.205375   0.9870    random-excurs-variant
  8   2   5  17   6   9   4  10   9   7  0.007616   0.9870    random-excurs-variant
  6   6   8   6  14   6   9  10   6   6  0.362174   0.9870    random-excurs-variant
  4   7  14   6  10   6  12   6   4   8  0.090936   0.9870    random-excurs-variant
  7   8   9   9   9  11   7   6   7   4  0.781926   1.0000    random-excurs-variant
  7   9  11   9  11   4   7   9   5   5  0.464055   1.0000    random-excurs-variant
 10   8  10   2   7   9   8  11   5   7  0.362174   1.0000    random-excurs-variant
 10   9  10   5   9   4   5   8   8   9  0.637119   0.9870    random-excurs-variant
  3   7   6   8   9  12   7  10   8   7  0.519816   1.0000    random-excurs-variant
  4   6   8   8  12   8  10   9   3   9  0.339044   1.0000    random-excurs-variant
  3   8  16   7  10   7   8   8   2   8  0.022870   0.9870    random-excurs-variant
  5  13  10  10   7   5  10   7   4   6  0.238562   1.0000    random-excurs-variant
  6  11   9   9   8   7   7   7   5   8  0.881013   0.9870    random-excurs-variant
  6  10  10   8  11   2   8   6  10   6  0.316916   0.9870    random-excurs-variant
  5   9  10  10   8   6  10   7   6   6  0.781926   0.9870    random-excurs-variant
  6  11   5   9   4  10  12  10   4   6  0.190212   0.9740    random-excurs-variant
  8   7   9   9   5   6   6  15   7   5  0.221483   1.0000    random-excurs-variant
 15   8   7  12  10  10   5  15  11   7  0.334538   0.9900    serial
 11   6  13   7  10  11  12  13  10   7  0.759756   0.9700    serial
  7  14  15   9   8  18   6  11   8   4  0.040108   1.0000    linear-complexity
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the random
excursion (variant) test is approximately = 0.960150 for a sample size = 100
binary sequences.

The minimum pass rate for the random excursion (variant) test is approximately
0.955983 for a sample size = 77 binary sequences.
```

ДОДАТОК 3. Програмний код шифру RSB-64-3D

Програмна реалізація шифру RSB-64-3D містить х класів:

1) “Form1.cs” – клас головного вікна програми в якому реалізовано графічну оболонку і функціонал;
2) “GISTO.cs” – клас, який вираховує значення ентропії і перетворює ключ введений користувачем у ключ, який використовується при шифруванні;
3) “MyBitA.cs” – клас, який займається перетворенням 1D, 2D, 3D структур на рвіні біт і SBox на основі адитивних компонентів;
4) “MyCrypt.cs” – клас, який є посередником між графічною оболонкою для користувача і криптографічними примітивами;
5) “MyStream.cs” – клас, який займається роботою з файлмаи, тобто зчитює файл відкритого тексту і формує на основі шифрограми файл з зашифрованим текстом;
6) “Permut_SUB_3D.cs” – клас, який реалізує криптографічний примітив перемішування і перемішування-підстановки;
7) “Permut_XYZ.cs” – клас, який реалізує криптографічний прмітив перемішування за напрямками;
8) “SlaideCode.cs” – клас, який реалізує криптографічний примітив «ковзючого кодування»;
9) “SubCode_3D.cs” – клас, який реалізує нелінійну заміну.

Програмний код головного вікна “Form1.cs”

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using System.IO;
using System.Collections; //для BitArray и Hashtable
using System.Security.Cryptography; //для RandomNumberGenerator.GetBytes - метод

namespace RSB_64_3D
{
```

```
    public partial class Form1 : Form
    {
        public Form_MO f_MO;

        const int kMax = 8; //размер кубика в байтах
        byte[] arrFileBytes; //байтовый массив данных из файла
        byte[] arrFileBytesEnd; //байтовый массив данных из файла
        List<bool[, ,]> listFile; //список 3D массивов данных из файла
        List<string[]> listTextKeyAll; //Список всех ключей в текстовом виде
        List<List<bool[, ,]>> listKeyAll; //Список всех ключей в виде списка массивов 3D
Bool
        //Для Permut_123_XYZ_3D
        List<string[]> listTextKeyAllX;
        List<string[]> listTextKeyAllY;
        List<string[]> listTextKeyAllZ;
        //Для Permut_Sub_Alfa_vs_Beta
        List<byte[]> listByte_Alfa, listByte_Beta;
        //Для Sub
        List<byte[]> listByte_Alfa_A, listByte_Beta_A;
        List<byte[]> listByte_Alfa_B, listByte_Beta_B;
        List<byte[]> listByte_Alfa_C, listByte_Beta_C;

        bool bildTabl_All = false;
        //bool bildTabl_A = false;
        //bool bildTabl_B = false;
        //bool bildTabl_C = false;
        //bool bildTabl_A_vs_B = false;

        Stream myStreamSaveFile;

        DateTime time_Start;
        DateTime time_End;
        TimeSpan time_My;

        public Form1()
        {
            InitializeComponent();
            comboBox_step.SelectedIndex = 0;
            comboBox_round.SelectedIndex = 0;
            //comboBox_round.SelectedIndex = 0; //количество раундов 1
            comboBox_block.SelectedIndex = 0; //в блоке 64 бита (1-н кубик)
            tabControl1.SelectedIndex = 3;
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            dGV_KeyMatrix_A.ColumnCount = dGV_KeyMatrix_B.ColumnCount =
dGV_KeyMatrix_C.ColumnCount = 8;
            dGV_KeyMatrix_A.RowCount = dGV_KeyMatrix_B.RowCount = dGV_KeyMatrix_C.RowCount =
8;

            tabl_KeyMatrix_A(); tabl_KeyMatrix_B(); tabl_KeyMatrix_C(); //Заполняем матрицу
"по умолчанию"

            cB_polynomial_KeyMatrix_A_1.SelectedIndex = 2;  //100101011
            cB_polynomial_KeyMatrix_B_1.SelectedIndex = 11; //101110001
            cB_polynomial_KeyMatrix_C_1.SelectedIndex = 21; //110111101
            cB_polynomial_KeyMatrix_A_2.SelectedIndex = 6;  //101001101
            cB_polynomial_KeyMatrix_B_2.SelectedIndex = 18; //110100011
            cB_polynomial_KeyMatrix_C_2.SelectedIndex = 12; //101110111
            cB_element_KeyMatrix_A.SelectedIndex = cB_element_KeyMatrix_B.SelectedIndex =
cB_element_KeyMatrix_C.SelectedIndex = 0;
```

```
            OB_A('0'); OB_B('0'); OB_C('0'); //OB_A(); OB_B(); OB_C(); //запись образующих
элементов

            //cB_XYZ_A.SelectedIndex = 1; //выбор напраления X
            //cB_XYZ_B.SelectedIndex = 2; //выбор напраления Y
            //cB_XYZ_C.SelectedIndex = 3; //выбор напраления Z
            cB_Sub_XYZ.SelectedIndex = 0; //выбор напраления X - начальное значение Sub
            cB_Sub_123.SelectedIndex = 2; //XYZ

            ///
            dGV_KeyMatrix_A_vs_B.ColumnCount = 6;
            dGV_KeyMatrix_A_vs_B.RowCount = 6;
            dGV_KeyMatrix_A_vs_B.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;

            tabl_KeyMatrix(); //Заполняем матрицу "по умолчанию"

            cB_polynomial_KeyMatrix_A_vs_B_1.SelectedIndex = 6; //1101101
            cB_polynomial_KeyMatrix_A_vs_B_2.SelectedIndex = 3; //1011011
            cB_element_KeyMatrix_A_vs_B.SelectedIndex = 0;

            OB_A_vs_B('0'); //OB_A_vs_B(); //запись образующих элементов

            cB_Perm_Sub.SelectedIndex = 0; //выбор напраления X
            cBox_Alfa.SelectedIndex = cBox_Alfa_A.SelectedIndex = cBox_Alfa_B.SelectedIndex
= cBox_Alfa_C.SelectedIndex = 0; //начальное значение Альфа
            cBox_Beta.SelectedIndex = cBox_Beta_A.SelectedIndex = cBox_Beta_B.SelectedIndex
= cBox_Beta_C.SelectedIndex = 0; //начальное значение Бетта
            ///
            //cB_Perm_XYZ_1.SelectedIndex = 1;
            //cB_Perm_XYZ_2.SelectedIndex = 2;
            //cB_Perm_XYZ_3.SelectedIndex = 3;
            cB_Perm_Sub.SelectedIndex = 0; //X
            cB_Perm_XYZ.SelectedIndex = 0; //X
            cB_Perm_123.SelectedIndex = 2; //XYZ
        }

        private void cB_polynomial_KeyMatrix_A_DropDownClosed(object sender, EventArgs e)
        {
            //OB_A(); //запись образующих элементов
        }

        private void cB_polynomial_KeyMatrix_A_2_DropDownClosed(object sender, EventArgs e)
        {
            //OB_A(); //запись образующих элементов
        }

        private void cB_element_KeyMatrix_A_DropDownClosed(object sender, EventArgs e)
        {
            //OB_A('w'); //запись матрицы элементов
        }

        private void cB_polynomial_KeyMatrix_B_DropDownClosed(object sender, EventArgs e)
        {
            //OB_B(); //запись образующих элементов
        }

        private void cB_polynomial_KeyMatrix_B_2_DropDownClosed(object sender, EventArgs e)
        {
            //OB_B(); //запись образующих элементов
        }

        private void cB_element_KeyMatrix_B_DropDownClosed(object sender, EventArgs e)
```

```
        {
            //OB_B('w'); //запись матрицы элементов
        }

        private void cB_polynomial_KeyMatrix_C_DropDownClosed(object sender, EventArgs e)
        {
            //OB_C(); //запись образующих элементов
        }

        private void cB_polynomial_KeyMatrix_C_2_DropDownClosed(object sender, EventArgs e)
        {
            //OB_C(); //запись образующих элементов
        }

        private void cB_element_KeyMatrix_C_DropDownClosed(object sender, EventArgs e)
        {
            //OB_C('w'); //запись матрицы элементов
        }

        private void cB_polynomial_KeyMatrix_A_vs_B_DropDownClosed(object sender, EventArgs
e)
        {
            //OB_A_vs_B();
        }

        private void cB_polynomial_KeyMatrix_A_vs_B_2_DropDownClosed(object sender,
EventArgs e)
        {
            //OB_A_vs_B();
        }

        private void cB_element_KeyMatrix_A_vs_B_DropDownClosed(object sender, EventArgs e)
        {
            //OB_A_vs_B('w');
        }

        private void checkBox_tabl_all_CheckedChanged(object sender, EventArgs e)
        {

        }

        private void Button_in_Click(object sender, EventArgs e)
        {
            arrFileBytes = MyStream.File_in; //массив байт из файла
            TextBox_in.Text = MyStream.strFileOpen; //название файла на интерфейсе

            try
            {
                double gistoVal_in = GISTO.start(arrFileBytes);
                Label_gisto_in.Text = "вх. файл\n\rH = " + gistoVal_in.ToString();
            }
            catch (Exception)
            {
                return;
                throw;
            }
        }

        private void Button_out_Click(object sender, EventArgs e)
        {
            myStreamSaveFile = MyStream.File_out;
            TextBox_out.Text = MyStream.strFileSave;
```

```
            Label_gisto_out.Text = "вых. файл\n\rH = пока неизвестна";
        }

        private void button_ok_Click(object sender, EventArgs e)
        {
            if (!File.Exists(TextBox_in.Text) || !File.Exists(TextBox_out.Text))
            {
                MessageBox.Show("Не указан файл", "Предупреждение", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }
            //проверка ключа
            if (Test_text(textBox_Key) != "OK")
            {
                cB_hand.Checked = true;
                button_ok.Enabled = false;
                return;
            }
            //задаем ключ
            modifierMasterKey();
            modifierPermutKey();
            //задаем Альфа и Бета
            modifierPermutSubKey();

            //выбираем количество шагов
            int step = comboBox_step.SelectedIndex; //начинаем с нулевого раунда
            int round = textBox_Key.Lines.Length - 1;//количество раундов кратно 64 битам
ключа
            progressBar_step.Maximum = step + 1;
            progressBar_raund.Maximum = round + 1;

            //time_Start = DateTime.Now;
            //System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.WaitCursor;
            this.Cursor = System.Windows.Forms.Cursors.WaitCursor;
            this.Enabled = false;
            Application.DoEvents();
            //Выбор направления для Sub
            byte uArr1 = 0;
            byte uArr2 = 0;
            byte uArr3 = 0;
            //выбор направлений для Sub
            if (rB_Sub_XYZ.Checked)
                switch (cB_Sub_XYZ.SelectedItem.ToString())
                {
                    case "X":
                        uArr1 = 1; uArr2 = 0; uArr3 = 0;
                        break;
                    case "Y":
                        uArr1 = 2; uArr2 = 0; uArr3 = 0;
                        break;
                    case "Z":
                        uArr1 = 3; uArr2 = 0; uArr3 = 0;
                        break;
                    default:
                        uArr1 = 0; uArr2 = 0; uArr3 = 0; //отключен шифратор
                        break;
                }
            if (rB_Sub_123.Checked)
                switch (cB_Sub_123.SelectedItem.ToString())
                {
                    case "X→Y":
                        uArr1 = 1; uArr2 = 2; uArr3 = 0;
                        break;
```

```
                    case "X→Z":
                        uArr1 = 1; uArr2 = 3; uArr3 = 0;
                        break;
                    case "Y→Z":
                        uArr1 = 2; uArr2 = 3; uArr3 = 0;
                        break;
                    case "X→Y→Z":
                        uArr1 = 1; uArr2 = 2; uArr3 = 3;
                        break;
                    default:
                        uArr1 = 0; uArr2 = 0; uArr3 = 0; //отключен шифратор
                        break;
                }

            //Подготовка таблицы MO, SBox, Reverse_SBox
            byte[,] MyMO, MySBox, MyRSBos;
            //A
            List<List<byte[,]>> MyArrMO_2D_A = new List<List<byte[,]>>();
            List<List<byte[,]>> MyArrSBox_2D_A = new List<List<byte[,]>>(); //для зашифровки
            List<List<byte[,]>> MyArrReverseSBox_2D_A = new List<List<byte[,]>>(); //для
расшифровки
            int wA = Convert.ToInt32((string)cB_element_KeyMatrix_A.SelectedItem, 2);
//Преобразуем двоичное число (записанное в виде строки) в десятичное число
            int pA = Convert.ToInt32((string)cB_polynomial_KeyMatrix_A_1.SelectedItem, 2);
//неприводимый полином
            //B
            List<List<byte[,]>> MyArrMO_2D_B = new List<List<byte[,]>>();
            List<List<byte[,]>> MyArrSBox_2D_B = new List<List<byte[,]>>(); //для зашифровки
            List<List<byte[,]>> MyArrReverseSBox_2D_B = new List<List<byte[,]>>(); //для
расшифровки
            //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!+1
            int wB = Convert.ToInt32((string)cB_element_KeyMatrix_B.SelectedItem, 2)+1;
//Преобразуем двоичное число (записанное в виде строки) в десятичное число
            int pB = Convert.ToInt32((string)cB_polynomial_KeyMatrix_B_1.SelectedItem, 2);
//неприводимый полином
            //C
            List<List<byte[,]>> MyArrMO_2D_C = new List<List<byte[,]>>();
            List<List<byte[,]>> MyArrSBox_2D_C = new List<List<byte[,]>>(); //для зашифровки
            List<List<byte[,]>> MyArrReverseSBox_2D_C = new List<List<byte[,]>>(); //для
расшифровки
            int wC = Convert.ToInt32((string)cB_element_KeyMatrix_C.SelectedItem, 2);
//Преобразуем двоичное число (записанное в виде строки) в десятичное число
            int pC = Convert.ToInt32((string)cB_polynomial_KeyMatrix_C_1.SelectedItem, 2);
//неприводимый полином

            if ((uArr1 != 0) | (uArr2 != 0) | (uArr3 != 0)) //Sub A,B,C
            {
                //Подготовка таблиц MO, S-Box и Обратный S-Box. Для Sub примитива.
                //MyBitA.KeyMatrix2D = ConvertMatrixToArray('A'); //преобразование матрицы
A,B,C ("0" и "1") в массив масивов. Т.е. каждый столбик бит будет соответствовать какому-то
числу
                byte[][] KeyMatrix2D_A = ConvertMatrixToArray('A'); ;
                byte[][] KeyMatrix2D_B = ConvertMatrixToArray('B'); ;
                byte[][] KeyMatrix2D_C = ConvertMatrixToArray('C'); ;
                for (int i_step = 0; i_step <= step; i_step++)
                {
                    MyArrMO_2D_A.Add(new List<byte[,]>());
                    MyArrMO_2D_B.Add(new List<byte[,]>());
                    MyArrMO_2D_C.Add(new List<byte[,]>());

                    MyArrSBox_2D_A.Add(new List<byte[,]>());
                    MyArrReverseSBox_2D_A.Add(new List<byte[,]>());
                    MyArrSBox_2D_B.Add(new List<byte[,]>());
```

```
                    MyArrReverseSBox_2D_B.Add(new List<byte[,]>());
                    MyArrSBox_2D_C.Add(new List<byte[,]>());
                    MyArrReverseSBox_2D_C.Add(new List<byte[,]>());

                    for (int i_round = 0; i_round <= round; i_round++)
                    {
                        //A
                        MyMO = new byte[0xF + 1, 0xF + 1];
                        MySBox = new byte[0xF + 1, 0xF + 1];
                        MyRSBos = new byte[0xF + 1, 0xF + 1];

                        MyBitA.KeyMatrix2D = KeyMatrix2D_A;
                        MyBitA.key_Alfa = listByte_Alfa_A[i_step][i_round];
                        MyBitA.key_Beta = listByte_Beta_A[i_step][i_round];
                        MyBitA.Tabl(wA, pA, ref MyMO, ref MySBox, ref MyRSBos, bildTabl_All,
'A');

                        MyArrMO_2D_A[i_step].Add(MyMO);
                        MyArrSBox_2D_A[i_step].Add(MySBox);
                        MyArrReverseSBox_2D_A[i_step].Add(MyRSBos);

                        //B
                        MyMO = new byte[0xF + 1, 0xF + 1];
                        MySBox = new byte[0xF + 1, 0xF + 1];
                        MyRSBos = new byte[0xF + 1, 0xF + 1];

                        MyBitA.KeyMatrix2D = KeyMatrix2D_B;
                        MyBitA.key_Alfa = listByte_Alfa_B[i_step][i_round];
                        MyBitA.key_Beta = listByte_Beta_B[i_step][i_round];
                        MyBitA.Tabl(wB, pB, ref MyMO, ref MySBox, ref MyRSBos, bildTabl_All,
'B');

                        MyArrMO_2D_B[i_step].Add(MyMO);
                        MyArrSBox_2D_B[i_step].Add(MySBox);
                        MyArrReverseSBox_2D_B[i_step].Add(MyRSBos);

                        //C
                        MyMO = new byte[0xF + 1, 0xF + 1];
                        MySBox = new byte[0xF + 1, 0xF + 1];
                        MyRSBos = new byte[0xF + 1, 0xF + 1];

                        MyBitA.KeyMatrix2D = KeyMatrix2D_C;
                        MyBitA.key_Alfa = listByte_Alfa_C[i_step][i_round];
                        MyBitA.key_Beta = listByte_Beta_C[i_step][i_round];
                        MyBitA.Tabl(wC, pC, ref MyMO, ref MySBox, ref MyRSBos, bildTabl_All,
'C');

                        MyArrMO_2D_C[i_step].Add(MyMO);
                        MyArrSBox_2D_C[i_step].Add(MySBox);
                        MyArrReverseSBox_2D_C[i_step].Add(MyRSBos);
                    }
                }
            }

            /// Permut-Sub
            List<List<byte[,]>> MyArrMO_2D_A_vs_B = new List<List<byte[,]>>();
            List<List<byte[,]>> MyArrSBox_2D_A_vs_B = new List<List<byte[,]>>();
            List<List<byte[,]>> MyArrReverseSBox_2D_A_vs_B = new List<List<byte[,]>>();
            //Подготовка таблицы MO, SBox, Reverse_SBox !!!!!!!!!!!!!!!!!!!!!!!!!!!!!+1
            int wA_vs_B = Convert.ToInt32((string)cB_element_KeyMatrix_A_vs_B.SelectedItem,
2) + 1; //Преобразуем двоичное число (записанное в виде строки) в десятичное число
```

```
            int pA_vs_B =
Convert.ToInt32((string)cB_polynomial_KeyMatrix_A_vs_B_1.SelectedItem, 2); //неприводимый
полином
            //Подготовка таблиц MO, S-Box и Обратный S-Box. Для Sub примитива.
            MyBitA.KeyMatrix2D = ConvertMatrixToArray(kMax - 2); //для Альфа и Бетта.
Преобразование матрицы ("0" и "1") в массив масивов. Т.е. каждый столбик бит будет
соответствовать какому-то числу
            for (int i_step = 0; i_step <= step; i_step++)
            {
                MyArrMO_2D_A_vs_B.Add(new List<byte[,]>());
                MyArrSBox_2D_A_vs_B.Add(new List<byte[,]>());
                MyArrReverseSBox_2D_A_vs_B.Add(new List<byte[,]>());

                for (int i_round = 0; i_round <= round; i_round++)
                {
                    MyMO = new byte[kMax, kMax];
                    MySBox = new byte[kMax, kMax];
                    MyRSBos = new byte[kMax, kMax];

                    MyBitA.key_Alfa = listByte_Alfa[i_step][i_round];
                    MyBitA.key_Beta = listByte_Beta[i_step][i_round];
                    MyBitA.Tabl(wA_vs_B, pA_vs_B, ref MyMO, ref MySBox, ref MyRSBos,
bildTabl_All);

                    MyArrMO_2D_A_vs_B[i_step].Add(MyMO);
                    MyArrSBox_2D_A_vs_B[i_step].Add(MySBox);
                    MyArrReverseSBox_2D_A_vs_B[i_step].Add(MyRSBos);
                }
            }

            //Выбор направления для Permut-Sub
            byte uArr = 0;
            if (rB_Perm_Sub.Checked)
                switch (cB_Perm_Sub.SelectedItem.ToString())
                {
                    case "X":
                        uArr = 1;
                        break;
                    case "Y":
                        uArr = 2;
                        break;
                    case "Z":
                        uArr = 3;
                        break;
                    default:
                        uArr = 0;
                        break;
                }
            ///
            byte Perm_XYZ_1 = 0;
            byte Perm_XYZ_2 = 0;
            byte Perm_XYZ_3 = 0;
            //Выбираем направление для Permut
            if (rB_Perm_XYZ.Checked)
                switch (cB_Perm_XYZ.SelectedItem.ToString())
                {
                    case "X":
                        Perm_XYZ_1 = 1; Perm_XYZ_2 = 0; Perm_XYZ_3 = 0;
                        break;
                    case "Y":
                        Perm_XYZ_1 = 2; Perm_XYZ_2 = 0; Perm_XYZ_3 = 0;
                        break;
                    case "Z":
```

```
                        Perm_XYZ_1 = 3; Perm_XYZ_2 = 0; Perm_XYZ_3 = 0;
                        break;
                    default:
                        Perm_XYZ_1 = 0; Perm_XYZ_2 = 0; Perm_XYZ_3 = 0; //отключен шифратор
                        break;
                }
            if (rB_Perm_123.Checked)
                switch (cB_Perm_123.SelectedItem.ToString())
                {
                    case "X→Y":
                        Perm_XYZ_1 = 1; Perm_XYZ_2 = 2; Perm_XYZ_3 = 0;
                        break;
                    case "X→Z":
                        Perm_XYZ_1 = 1; Perm_XYZ_2 = 3; Perm_XYZ_3 = 0;
                        break;
                    case "Y→Z":
                        Perm_XYZ_1 = 2; Perm_XYZ_2 = 3; Perm_XYZ_3 = 0;
                        break;
                    case "X→Y→Z":
                        Perm_XYZ_1 = 1; Perm_XYZ_2 = 2; Perm_XYZ_3 = 3;
                        break;
                    default:
                        Perm_XYZ_1 = 0; Perm_XYZ_2 = 0; Perm_XYZ_3 = 0; //отключен шифратор
                        break;
                }


            int bMax = int.Parse((string)comboBox_block.SelectedItem) / kMax; //находим
размер кратный блоку
            int iMax = arrFileBytes.Length; //длина файла
            byte[] value = new byte[kMax]; //Байтовый массив кубика
            //Делаем массив файла кратным размеру кубика (64 бита или 8 байт)
            byte iChang = 0;
            if (radioButton_in.Checked)
            {
                if (iMax % bMax != 0) Array.Resize(ref arrFileBytes, (int)((int)(iMax /
bMax) + 1) * bMax);
                if (iMax % bMax == 0) Array.Resize(ref arrFileBytes,
(int)(arrFileBytes.Length + bMax));
                //записываем в последний байт количество "пустой" инфрмации
                iChang = (byte)(arrFileBytes.Length - iMax);
                arrFileBytes[arrFileBytes.Length - 1] = iChang;
            }

            //if (rB_B.Checked == true) if (iMax % bMax != 0) Array.Resize(ref arrFileBytes,
(int)((int)(iMax / bMax) + 1) * bMax);
            iMax = arrFileBytes.Length; //новая длина файла
            int blokCub = bMax / kMax; //количество кубиков в блоке
            int blokMax = iMax / bMax; //количество блоков в файле
            //Преобразовываем массив байт в массив Bool
            //Преобразование 1D массива Bool в коллекцию массивов 3D Bool
            listFile = ConvertByteToBoolList(arrFileBytes);

            bool cript_flag = radioButton_in.Checked == true ? true : false; //если
"зашифровать", то true. Если "расшифровать", то fals
            bool flag_A = rB_A.Checked;

            //выбираем подключаемые криптографические примитивы
            //bool flag_SlaideCode = cB_SlaideCode.Checked;
            //bool flag_Sub = cB_Sub.Checked;
            //bool flag_PermutSub = cB_PermutSub.Checked;
            //bool flag_Permut = cB_Permut.Checked;
```

```
            //string[] MyArrKey = textBox_Key.Lines;
            List<bool[, ,]> listKey; //список 3D массивов ключа

            time_Start = DateTime.Now;

            if (cript_flag) //"зашифровать"
            {
                for (int i_step = 0; i_step <= step; i_step++) //внешний цикл (меняется
ключ)
                {
                    //создаем новый ключ (модификация)
                    textBox_Key.Lines = listTextKeyAll[i_step]; //выводим ключ на форму
                    listKey = listKeyAll[i_step]; //задаем список 64-х битных ключей для
текущего шага
                    Application.DoEvents();
                    // средний цикл шифрования (раунды)
                    for (int i_round = 0; i_round <= round; i_round++)
                    {
                        //SlaideCode
                        MyCrypt.crypt_SlaideCode_in(ref listFile, i_round, listKey[i_round],
blokCub, flag_A); //"зашифровать"

                        //Sub
                        if (uArr1 != 0) MyCrypt.crypt_Sub_in(ref listFile,
MyArrSBox_2D_A[i_step][i_round], uArr1); //зашифровать
                        if (uArr2 != 0) MyCrypt.crypt_Sub_in(ref listFile,
MyArrSBox_2D_B[i_step][i_round], uArr2); //зашифровать
                        if (uArr3 != 0) MyCrypt.crypt_Sub_in(ref listFile,
MyArrSBox_2D_C[i_step][i_round], uArr3); //зашифровать

                        //PermutSub
                        if (uArr != 0) MyCrypt.crypt_Permut_SUB_in(ref listFile,
MyArrSBox_2D_A_vs_B[i_step][i_round], uArr); //зашифровать

                        //Permut
                        if (Perm_XYZ_1 != 0) MyCrypt.crypt_Permut_XYZ_in(ref listFile,
listTextKeyAllX[i_step][i_round], listTextKeyAllY[i_step][i_round],
listTextKeyAllZ[i_step][i_round], Perm_XYZ_1);
                        if (Perm_XYZ_2 != 0) MyCrypt.crypt_Permut_XYZ_in(ref listFile,
listTextKeyAllX[i_step][i_round], listTextKeyAllY[i_step][i_round],
listTextKeyAllZ[i_step][i_round], Perm_XYZ_2);
                        if (Perm_XYZ_3 != 0) MyCrypt.crypt_Permut_XYZ_in(ref listFile,
listTextKeyAllX[i_step][i_round], listTextKeyAllY[i_step][i_round],
listTextKeyAllZ[i_step][i_round], Perm_XYZ_3);

                        progressBar_raund.Value = i_round + 1;
                        label_raund.Text = (i_round + 1).ToString();
                        Application.DoEvents();
                    }
                    progressBar_step.Value = i_step + 1;
                    label_step.Text = (i_step + 1).ToString();
                }
            }
            else //"расшифровать"
            {
                for (int i_step = step; i_step >= 0; i_step--) //внешний цикл (меняется
ключ)
                {
                    //создаем новый ключ (модификация)
                    textBox_Key.Lines = listTextKeyAll[i_step]; //выводим ключ на форму
                    listKey = listKeyAll[i_step]; //задаем список 64-х битных ключей для
текущего шага
                    Application.DoEvents();
```

```
                    //// средний цикл шифрования (раунды)
                    for (int i_round = round; i_round >= 0; i_round--)
                    {
                        //Permut
                        if (Perm_XYZ_3 != 0) MyCrypt.crypt_Permut_XYZ_out(ref listFile,
listTextKeyAllX[i_step][i_round], listTextKeyAllY[i_step][i_round],
listTextKeyAllZ[i_step][i_round], Perm_XYZ_3);
                        if (Perm_XYZ_2 != 0) MyCrypt.crypt_Permut_XYZ_out(ref listFile,
listTextKeyAllX[i_step][i_round], listTextKeyAllY[i_step][i_round],
listTextKeyAllZ[i_step][i_round], Perm_XYZ_2);
                        if (Perm_XYZ_1 != 0) MyCrypt.crypt_Permut_XYZ_out(ref listFile,
listTextKeyAllX[i_step][i_round], listTextKeyAllY[i_step][i_round],
listTextKeyAllZ[i_step][i_round], Perm_XYZ_1);

                        //PermutSub
                        if (uArr != 0) MyCrypt.crypt_Permut_SUB_out(ref listFile,
MyArrReverseSBox_2D_A_vs_B[i_step][i_round], uArr); //расшифровать

                        //Sub
                        if (uArr3 != 0) MyCrypt.crypt_Sub_out(ref listFile,
MyArrReverseSBox_2D_C[i_step][i_round], uArr3); //расшифровать
                        if (uArr2 != 0) MyCrypt.crypt_Sub_out(ref listFile,
MyArrReverseSBox_2D_B[i_step][i_round], uArr2); //расшифровать
                        if (uArr1 != 0) MyCrypt.crypt_Sub_out(ref listFile,
MyArrReverseSBox_2D_A[i_step][i_round], uArr1); //расшифровать

                        //SlaideCode
                        MyCrypt.crypt_SlaideCode_out(ref listFile, i_round,
listKey[i_round], blokCub, flag_A); //"расшифровать"

                        progressBar_raund.Value = i_round + 1;
                        label_raund.Text = (i_round).ToString();
                        Application.DoEvents();
                    }
                    progressBar_step.Value = i_step + 1;
                    label_step.Text = (i_step).ToString();
                }
            }

            time_End = DateTime.Now;

            textBox_Key.Lines = listTextKeyAll[0]; //выводим первоначальный ключ на форму
            TextBoxX.Lines = listTextKeyAllX[0];
            TextBoxY.Lines = listTextKeyAllY[0];
            TextBoxZ.Lines = listTextKeyAllZ[0];
            if (bildTabl_All)
            {
                //меняем на интерфейсе Альфа и Бета
                Changes_Alfa_vs_Beta(0, 0);
            }

            //преобразовываем массив бит в массив байт (для файла)
            arrFileBytesEnd = ConvertBoolToByte(listFile);

            if (radioButton_out.Checked)
            {   //извлекаем данные о количестве "пустой" информации
                iChang = arrFileBytes[arrFileBytesEnd.Length - 1]; //iChang =
arrFileBytesEnd[arrFileBytesEnd.Length - 1];
                Array.Resize(ref arrFileBytesEnd, iMax - iChang);
            }
            iMax = arrFileBytesEnd.Length; //новая длина файла

            //записываем массив в файл
```

```
            myStreamSaveFile.Write(arrFileBytesEnd, 0, iMax);
            myStreamSaveFile.Close();

            //time_End = DateTime.Now;

            time_My = time_End - time_Start;
            label_Time.Text = "Время счета ( мин.: сек.): " + time_My;

            double gistoVal_out = GISTO.start(arrFileBytesEnd);
            Label_gisto_out.Text = "вых. файл\n\rH = " + gistoVal_out.ToString();

            this.Cursor = System.Windows.Forms.Cursors.Default;
            //System.Windows.Forms.Cursor.Current = System.Windows.Forms.Cursors.Default;
            progressBar_step.Value = progressBar_raund.Value = 0;
            this.Enabled = true;

            ///
            if (f_MO != null)
            {
                f_MO.Close();
                f_MO = null;
            }
            //строим таблицу
            if (bildTabl_All) //Если есть данные для таблицы, то:
            {
                //Form_MO f_MO = new Form_MO();
                if (f_MO == null)
                {
                    f_MO = new Form_MO();
                }
                f_MO.round = comboBox_round.SelectedIndex + 1; //передаем в форму
максимальное количество раундов для ключа
                f_MO.step = comboBox_step.SelectedIndex + 1; //передаем в форму максимальное
количество шагов для ключа

                f_MO.MyArrMO_2D_A = MyArrMO_2D_A;
                f_MO.MyArrSBox_2D_A = MyArrSBox_2D_A; //для зашифровки
                f_MO.MyArrReverseSBox_2D_A = MyArrReverseSBox_2D_A; //для расшифровки

                f_MO.MyArrMO_2D_B = MyArrMO_2D_B;
                f_MO.MyArrSBox_2D_B = MyArrSBox_2D_B; //для зашифровки
                f_MO.MyArrReverseSBox_2D_B = MyArrReverseSBox_2D_B; //для расшифровки

                f_MO.MyArrMO_2D_C = MyArrMO_2D_C;
                f_MO.MyArrSBox_2D_C = MyArrSBox_2D_C; //для зашифровки
                f_MO.MyArrReverseSBox_2D_C = MyArrReverseSBox_2D_C; //для расшифровки

                f_MO.MyArrMO_2D_A_vs_B = MyArrMO_2D_A_vs_B;
                f_MO.MyArrSBox_2D_A_vs_B = MyArrSBox_2D_A_vs_B;
                f_MO.MyArrReverseSBox_2D_A_vs_B = MyArrReverseSBox_2D_A_vs_B;

                //Для Permut_Sub_Alfa_vs_Beta
                f_MO.listByte_Alfa = listByte_Alfa;
                f_MO.listByte_Beta = listByte_Beta;
                //Для Sub
                f_MO.listByte_Alfa_A = listByte_Alfa_A;
                f_MO.listByte_Beta_A = listByte_Beta_A;
                f_MO.listByte_Alfa_B = listByte_Alfa_B;
                f_MO.listByte_Beta_B = listByte_Beta_B;
                f_MO.listByte_Alfa_C = listByte_Alfa_C;
                f_MO.listByte_Beta_C = listByte_Beta_C;

                f_MO.listTextKeyAll = listTextKeyAll; //Список всех ключей в текстовом виде
```

```
            f_MO.Show();
        }
        ///
    }

    private void Changes_Alfa_vs_Beta(int i_step, int i_round)
    {
        //меняем на интерфейсе Альфа и Бета для Permut_Sub
        cBox_Alfa.SelectedIndex = listByte_Alfa[i_step][i_round];
        cBox_Beta.SelectedIndex = listByte_Alfa[i_step][i_round];
        //меняем на интерфейсе Альфа и Бета для Sub
        cBox_Alfa_A.SelectedIndex = listByte_Alfa_A[i_step][i_round];
        cBox_Alfa_B.SelectedIndex = listByte_Alfa_B[i_step][i_round];
        cBox_Alfa_C.SelectedIndex = listByte_Alfa_C[i_step][i_round];

        cBox_Beta_A.SelectedIndex = listByte_Beta_A[i_step][i_round];
        cBox_Beta_B.SelectedIndex = listByte_Beta_B[i_step][i_round];
        cBox_Beta_C.SelectedIndex = listByte_Beta_C[i_step][i_round];
    }

    private void modifierMasterKey()
    {
        int step = comboBox_step.SelectedIndex; //начинаем с нулевого шага
        int round = textBox_Key.Lines.Length - 1; // comboBox_round.SelectedIndex; ;
//начинаем с нулевого раунда textBox_Key.Lines.Length
        int j = 0;
        //задаем ключ
        string[] MyArrKeyMaster = textBox_Key.Lines;
        string[] MyArrKeyStep;
        byte[] MyArrKeyRaundByte = new Byte[kMax * (round + 1)]; //ключ для одного шага
в байтах
        /*List<string[]>*/ listTextKeyAll = new List<string[]>(); //ключи для
наглядности
        List<bool[, ,]> listKeyStep; //список 3D массивов ключа для одного шага
(модификация первого ключа)
        /*List<List<bool[, ,]>>*/ listKeyAll = new List<List<bool[, ,]>>(); // список
всех модификаций первого ключа

        for (int i_step = 0; i_step <= step; i_step++)
        {
            MyArrKeyStep = GISTO.MyArrKeyF(MyArrKeyMaster, i_step); //берем ключ
            listTextKeyAll.Add(MyArrKeyStep); //список всех ключей в текстовом виде

            j = 0;
            for (int i_raund = 0; i_raund <= round; i_raund++)
            {
                for (int i = 0; i < 2*kMax; i += 2)
                {
                    MyArrKeyRaundByte[j++] =
Convert.ToByte(MyArrKeyStep[i_raund].Substring(i, 2), 16);
                }
            }

            listKeyStep = ConvertByteToBoolList(MyArrKeyRaundByte); //Преобразование 1D
массива Bool в массив массивов 3D массивов Bool
            listKeyAll.Add(listKeyStep); //список всех ключей в виде списков содержащих
3D Bool ключи
        }
    }

    private void cB_hand_CheckedChanged(object sender, EventArgs e)
    {
```

```
            if (cB_hand.Checked)
            {
                textBox_Key.ReadOnly = false;
                button_ok.Enabled = false;
            }
            else
            {
                if (Test_text(textBox_Key) != "OK")
                {
                    cB_hand.Checked = true;
                    return;
                }

                key_XYZ();

                textBox_Key.ReadOnly = true;
                button_ok.Enabled = true;
            }
        }

        private void button_Key_Click(object sender, EventArgs e)
        {
            //гениратор случайных чисел
            int round = comboBox_round.SelectedIndex + 1; //количество раундов
            byte[] randomKey = new Byte[kMax * round]; //полный ключ в байтах
            //RNGCryptoServiceProvider is an implementation of a random number generator.
            RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
            rng.GetBytes(randomKey); // The array is now filled with cryptographically
strong random bytes.
            string strKey = BitConverter.ToString(randomKey).Replace("-", string.Empty);

            string[] Text_Key = new string[round];
            for (int index = 0; index < round; index++)
            {
                Text_Key[index] = strKey.Substring(index * (2 * kMax), 2 * kMax);
            }
            textBox_Key.Lines = Text_Key;

            key_XYZ(); //для Пермута (для оси X,Y,Z) создаем ключ???
        }

        private void key_XYZ() //преобразование общего ключа в ключи для Permut XYZ 3D
        {
            string[] MyArrKey = textBox_Key.Lines;
            int MyArrKey_Length = MyArrKey.Length;
            string[] MyArrKeyX = new string[MyArrKey_Length];
            string[] MyArrKeyY = new string[MyArrKey_Length];
            string[] MyArrKeyZ = new string[MyArrKey_Length];

            for (int i = 0; i < MyArrKey_Length; i++)
            {
                MyArrKeyX[i] = MyArrKey[i].Remove(kMax, kMax);
                MyArrKeyY[i] = MyArrKey[i].Remove(0, kMax);
            }
            TextBoxX.Lines = MyArrKeyX;
            TextBoxY.Lines = MyArrKeyY;
            TextBoxZ.Lines = Permut_XYZ.MyArrKeyF(MyArrKey); //берем половину ключа
        }

        private void modifierPermutKey()
        {
            int step = comboBox_step.SelectedIndex; //начинаем с нулевого шага
```

```
            int round = textBox_Key.Lines.Length - 1; //количество раундов кратно 64 битам
ключа
            listTextKeyAllX = new List<string[]>();
            listTextKeyAllY = new List<string[]>();
            listTextKeyAllZ = new List<string[]>();
            //задаем ключ
            string[] MyArrKeyX;
            string[] MyArrKeyY;
            string[] MyArrKeyZ;

            //byte[] MyArrKeyRaundByte = new Byte[kMax * (step + 1)]; //ключ для одного шага
в байтах

            for (int i_step = 0; i_step <= step; i_step++)
            {
                MyArrKeyX = new string[round + 1];
                MyArrKeyY = new string[round + 1];
                MyArrKeyZ = new string[round + 1];

                for (int i = 0; i <= round; i++)
                {
                    MyArrKeyX[i] = listTextKeyAll[i_step][i].Remove(kMax, kMax);
                    MyArrKeyY[i] = listTextKeyAll[i_step][i].Remove(0, kMax);
                }
                listTextKeyAllX.Add(MyArrKeyX); //список всех ключей в текстовом виде
                listTextKeyAllY.Add(MyArrKeyY); //список всех ключей в текстовом виде

                MyArrKeyZ = Permut_XYZ.MyArrKeyF(listTextKeyAll[i_step]); //берем ключ
                listTextKeyAllZ.Add(MyArrKeyZ); //список всех ключей в текстовом виде
            }
        }

        private void modifierPermutSubKey()
        { //Находим все Альфы и Беты для Sub (A,B,C) и Permut-Sub
            int step = comboBox_step.SelectedIndex; //начинаем с нулевого шага
            int round = textBox_Key.Lines.Length - 1; //количество раундов кратно 64 битам
ключа
            listByte_Alfa = new List<byte[]>(); listByte_Beta = new List<byte[]>();
            listByte_Alfa_A = new List<byte[]>(); listByte_Beta_A = new List<byte[]>();
            listByte_Alfa_B = new List<byte[]>(); listByte_Beta_B = new List<byte[]>();
            listByte_Alfa_C = new List<byte[]>(); listByte_Beta_C = new List<byte[]>();
            //задаем ключ
            byte[] MyArrKey_Alfa, MyArrKey_Beta;
            byte[] MyArrKey_Alfa_A, MyArrKey_Alfa_B, MyArrKey_Alfa_C, MyArrKey_Beta_A,
MyArrKey_Beta_B, MyArrKey_Beta_C;

            for (int i_step = 0; i_step <= step; i_step++)
            {
                MyArrKey_Alfa = new byte[round + 1]; MyArrKey_Beta = new byte[round + 1];
                MyArrKey_Alfa_A = new byte[round + 1]; MyArrKey_Alfa_B = new byte[round +
1]; MyArrKey_Alfa_C = new byte[round + 1];
                MyArrKey_Beta_A = new byte[round + 1]; MyArrKey_Beta_B = new byte[round +
1]; MyArrKey_Beta_C = new byte[round + 1];

                for (int i = 0; i <= round; i++)
                {
                    MyArrKey_Alfa_A[i] =
Convert.ToByte(listTextKeyAll[i_step][i].Substring(0, 2), 16); //копируем первые 2 буквы для
Альфа и Бета Матрицы А
                    MyArrKey_Alfa_B[i] =
Convert.ToByte(listTextKeyAll[i_step][i].Substring(2, 2), 16); //копируем вторые 2 буквы для
Альфа и Бета Матрицы В
```

```
                    MyArrKey_Alfa_C[i] =
Convert.ToByte(listTextKeyAll[i_step][i].Substring(4, 2), 16); //копируем третьи 2 буквы для
Альфа и Бета Матрицы С

                    MyArrKey_Beta_A[i] =
Convert.ToByte(listTextKeyAll[i_step][i].Substring(6, 2), 16); //копируем первые 2 буквы для
Альфа и Бета Матрицы А
                    MyArrKey_Beta_B[i] =
Convert.ToByte(listTextKeyAll[i_step][i].Substring(8, 2), 16); //копируем вторые 2 буквы для
Альфа и Бета Матрицы В
                    MyArrKey_Beta_C[i] =
Convert.ToByte(listTextKeyAll[i_step][i].Substring(10,2), 16); //копируем третьи 2 буквы для
Альфа и Бета Матрицы С

                    MyArrKey_Alfa[i] =
(byte)(Convert.ToByte(listTextKeyAll[i_step][i].Substring(12, 2), 16) >> 2); //копируем
третьи 2 буквы для Альфа и Бета Матрицы С
                    MyArrKey_Beta[i] =
(byte)(Convert.ToByte(listTextKeyAll[i_step][i].Substring(14, 2), 16) >> 2); //копируем
третьи 2 буквы для Альфа и Бета Матрицы С

                }
                listByte_Alfa_A.Add(MyArrKey_Alfa_A);
                listByte_Alfa_B.Add(MyArrKey_Alfa_B);
                listByte_Alfa_C.Add(MyArrKey_Alfa_C);

                listByte_Beta_A.Add(MyArrKey_Beta_A);
                listByte_Beta_B.Add(MyArrKey_Beta_B);
                listByte_Beta_C.Add(MyArrKey_Beta_C);

                listByte_Alfa.Add(MyArrKey_Alfa);
                listByte_Beta.Add(MyArrKey_Beta);
            }

        }

        private void button_Open_Key_Click(object sender, EventArgs e)
        {//Открытие ключа
            byte[] arrKeyBytes = null; //байтовый массив данных из файла ключа
            if ((arrKeyBytes = MyStream.File_in) != null)
            { //массив байт из файла ключа
                string strKey = "";
                int multiple = arrKeyBytes.Length / (2 * kMax); //сколько строк в ключе
                textBox_Key.Text = "";

                for (int i = 0; i < multiple; i++)
                {
                    strKey = "";
                    for (int item = 0; item < 2*kMax; item++)
                    {
                        strKey += (char) arrKeyBytes[i*(2*kMax) + item];
                    }
                    if (i == 0) textBox_Key.Text = strKey;
                    else textBox_Key.Text = textBox_Key.Text + "\r\n" + strKey;
                }
                comboBox_round.SelectedIndex = multiple - 1;
                //foreach (byte item in arrKeyBytes) { strKey += (char)item; }
                //textBox_Key.Text = strKey;
            }
            key_XYZ();
        }

        private void button_Save_Key_Click(object sender, EventArgs e)
```

```
        {//Сохранение ключа
            Stream myStreamSaveKey = null;
            if ((myStreamSaveKey = MyStream.File_out) != null)
            {
                byte[] b = new byte[textBox_Key.TextLength]; //полная длинна ключа
                int num = 0;
                string strKey = "";
                int multiple = textBox_Key.Lines.Length; //сколько строк в ключе
                for (int i = 0; i < multiple; i++)
                {
                    strKey = textBox_Key.Lines[i];
                    for (int j = 0; j < 2*kMax; j++)
                    {
                        b[num++] = (byte) strKey[j];
                    }
                }
                myStreamSaveKey.Write(b, 0, num);
                myStreamSaveKey.Close();
            }

        }

        List<bool[, ,]> ConvertByteToBoolList(byte[] random)
        {
            //Преобразовываем массив байт в массив Bool
            BitArray arrBool = new BitArray(random);
            //Преобразование 1D массива Bool в массив массивов 3D массивов Bool
            int xyzMax = (int)Math.Round(Math.Pow(kMax * kMax, 1.0 / 3.0), 0); //размер
полубайта 4 )))
            int cub = (int)Math.Pow(xyzMax, 3.0); //размер кубика
            int xyz = 0;
            int BoolMax = arrBool.Length / cub;
            bool[, ,] arrBool_3D = null;// = new bool[xyzMax, xyzMax, xyzMax];
            List<bool[, ,]> Listes = new List<bool[, ,]>();
            for (int i = 0; i < BoolMax; i++)
            {
                arrBool_3D = new bool[xyzMax, xyzMax, xyzMax];
                for (int z = 0; z < xyzMax; z++)
                {
                    for (int y = 0; y < xyzMax; y++)
                    {
                        for (int x = 0; x < xyzMax; x++)
                        {
                            arrBool_3D[z, y, x] = arrBool[xyz++];
                        }
                    }
                }
                Listes.Add(arrBool_3D);
            }
            return Listes;
        }

        byte[] ConvertBoolToByte(List<bool[, ,]> Listes)
        {
            int xyzMax = (int)Math.Round(Math.Pow(kMax * kMax, 1.0 / 3.0), 0); //размер
полубайта 4 )))
            bool[, ,] arrBool_3D = null;
            BitArray arrBitBool = new BitArray(64);
            int BoolMax = Listes.Count;
            byte[] arrBytes = new byte[kMax];
            int g = 0;
            byte[] ret = new byte[8];
            for (int i = 0; i < BoolMax; i++, g = 0)
```

```
            {
                arrBool_3D = Listes[i];
                //запись в одномерный массив bool
                for (int z = 0; z < xyzMax; z++)
                {
                    for (int y = 0; y < xyzMax; y++)
                    {
                        for (int x = 0; x < xyzMax; x++)
                        {
                            //arrBool[h++] = arrBool_3D[z, y, x];
                            arrBitBool[g++] = arrBool_3D[z, y, x];
                        }
                    }
                }

                //преобразование bool в byte
                arrBitBool.CopyTo(ret, 0);
                Array.Copy(ret, 0, arrFileBytes, i * kMax, kMax);
            }
            return arrFileBytes;
        }

        byte[][] ConvertMatrixToArray(char xD)
        {
            //преобразование матрицы ("0" и "1") в массив масивов. Т.е. каждый столбик бит
будет соответствовать какому-то числу
            BitArray arrBitBool = new BitArray(kMax);
            byte[][] ret = new byte[kMax][]; //файктически это массив столбиков. С
преобразоваными битами в числа int
            for (int x = 0; x < kMax; x++)
            {
                for (int y = 0; y < kMax; y++)
                {
                    switch (xD)
                    {
                        case 'A':
                            arrBitBool[y] =
Convert.ToBoolean(Convert.ToByte(dGV_KeyMatrix_A[x, kMax - 1 - y].Value)); //вытягиваем
значение из матрицы и преобразовываем его в логическое значение
                            break;
                        case 'B':
                            arrBitBool[y] =
Convert.ToBoolean(Convert.ToByte(dGV_KeyMatrix_B[x, kMax - 1 - y].Value)); //вытягиваем
значение из матрицы и преобразовываем его в логическое значение
                            break;
                        case 'C':
                            arrBitBool[y] =
Convert.ToBoolean(Convert.ToByte(dGV_KeyMatrix_C[x, kMax - 1 - y].Value)); //вытягиваем
значение из матрицы и преобразовываем его в логическое значение
                            break;
                        default:
                            break;
                    }
                }
                ret[x] = new byte[1]; //создаем массив для одного числа (которое получится
из массива бит для одного столбика
                arrBitBool.CopyTo(ret[x], 0); //запись логического массива (биты) в массив
чисел
            }
            return ret;
        }

        byte[][] ConvertMatrixToArray(int kkMax = kMax)
```

```
        {
            //преобразование матрицы ("0" и "1") в массив масивов. Т.е. каждый столбик бит
будет соответствовать какому-то числу
            BitArray arrBitBool = new BitArray(kkMax);
            byte[][] ret = new byte[kkMax][]; //файктически это массив столбиков. С
преобразоваными битами в числа int
            for (int x = 0; x < kkMax; x++)
            {
                for (int y = 0; y < kkMax; y++)
                {
                    arrBitBool[y] = Convert.ToBoolean(Convert.ToByte(dGV_KeyMatrix_A_vs_B[x,
kkMax - 1 - y].Value)); //вытягиваем значение из матрицы и преобразовываем его в логическое
значение
                }
                ret[x] = new byte[1]; //создаем массив для одного числа (которое получится
из массива бит для одного столбика
                arrBitBool.CopyTo(ret[x], 0); //запись логического массива (биты) в массив
чисел
            }
            return ret;
        }

        void tabl_KeyMatrix_A()
        {
            //Заполняем матрицу "по умолчанию"
            for (int y = 0; y < 8; y++)
            {
                for (int x = 0; x < 8; x++)
                {
                    if (x == y)
                    {
                        dGV_KeyMatrix_A[x, y].Value = 1;
                        dGV_KeyMatrix_A[x, y].Style.BackColor = SystemColors.ControlLight;
                    }
                    else dGV_KeyMatrix_A[x, y].Value = 0;
                }
                dGV_KeyMatrix_A.Rows[y].Height = dGV_KeyMatrix_A.Columns[y].Width = 20;
            }
        }

        void tabl_KeyMatrix_B()
        {
            //Заполняем матрицу "по умолчанию"
            for (int y = 0; y < 8; y++)
            {
                for (int x = 0; x < 8; x++)
                {
                    if (x == y)
                    {
                        dGV_KeyMatrix_B[x, y].Value = 1;
                        dGV_KeyMatrix_B[x, y].Style.BackColor = SystemColors.ControlLight;
                    }
                    else dGV_KeyMatrix_B[x, y].Value = 0;
                }
                dGV_KeyMatrix_B.Rows[y].Height = dGV_KeyMatrix_B.Columns[y].Width = 20;
            }
        }

        void tabl_KeyMatrix_C()
        {
            //Заполняем матрицу "по умолчанию"
            for (int y = 0; y < 8; y++)
            {
```

```
                for (int x = 0; x < 8; x++)
                {
                    if (x == y)
                    {
                        dGV_KeyMatrix_C[x, y].Value = 1;
                        dGV_KeyMatrix_C[x, y].Style.BackColor = SystemColors.ControlLight;
                    }
                    else dGV_KeyMatrix_C[x, y].Value = 0;
                }
                dGV_KeyMatrix_C.Rows[y].Height = dGV_KeyMatrix_C.Columns[y].Width = 20;
            }
        }

        void tabl_KeyMatrix()
        {
            //Заполняем матрицу "по умолчанию"
            for (int y = 0; y < 6; y++)
            {
                for (int x = 0; x < 6; x++)
                {
                    if (x == y) { dGV_KeyMatrix_A_vs_B[x, y].Value = 1;
dGV_KeyMatrix_A_vs_B[x, y].Style.BackColor = SystemColors.ControlLight; }
                    else dGV_KeyMatrix_A_vs_B[x, y].Value = 0;
                }
                dGV_KeyMatrix_A_vs_B.Rows[y].Height = dGV_KeyMatrix_A_vs_B.Columns[y].Width
= 20;
            }
        }

        void OB_A_vs_B(char pw = 'p')
        {   //НП
            string p_KeyMatrixStr =
cB_polynomial_KeyMatrix_A_vs_B_2.SelectedItem.ToString(); //выбор неприводимого полинома
            int p_KeyMatrixInt = Convert.ToInt32(p_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            if (pw == 'p') cB_element_KeyMatrix_A_vs_B.DataSource =
MyBitA.bitModXor(p_KeyMatrixInt); //запись образующих элементов
            //ОЭ
            string w_KeyMatrixStr = cB_element_KeyMatrix_A_vs_B.SelectedItem.ToString();
//выбор неприводимого полинома
            int w_KeyMatrixInt = Convert.ToInt32(w_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            //Матрица
            tabl_KeyMatrix(p_KeyMatrixInt, w_KeyMatrixInt, 'v');
        }

        void tabl_KeyMatrix(int p, int w, char xD, int length = 6)
        {
            //Заполняем матрицу
            int w_new = w;
            int len = 0;
            if (length == 6) len = 0x40;
            if (length == 8) len = 0x100;
            int[] MyArr = new int[length];
            MyArr[0] = w_new;

            for (int i = 1; i < length; i++)
            {
                w_new = w_new << 1;
                if (w_new >= len) w_new = p ^ w_new; //Если выходит за пределы матрицы
                MyArr[i] = w_new;
            }
            len = len >> 1; length--; //для позиционирования числа в матрице
```

```
            for (int y = 0; y <= length; y++)
            {
                for (int x = 0; x <= length; x++)
                {
                    switch (xD)
                    {
                        case 'A':
                            if ((MyArr[y] & (len >> x)) != 0) { dGV_KeyMatrix_A[x, length -
y].Value = 1; dGV_KeyMatrix_A[x, length - y].Style.BackColor = SystemColors.ControlLight; }
                            else { dGV_KeyMatrix_A[x, length - y].Value = 0;
dGV_KeyMatrix_A[x, length - y].Style.BackColor = SystemColors.Window; }
                            break;
                        case 'B':
                            if ((MyArr[y] & (len >> x)) != 0) { dGV_KeyMatrix_B[x, length -
y].Value = 1; dGV_KeyMatrix_B[x, length - y].Style.BackColor = SystemColors.ControlLight; }
                            else { dGV_KeyMatrix_B[x, length - y].Value = 0;
dGV_KeyMatrix_B[x, length - y].Style.BackColor = SystemColors.Window; }
                            break;
                        case 'C':
                            if ((MyArr[y] & (len >> x)) != 0) { dGV_KeyMatrix_C[x, length -
y].Value = 1; dGV_KeyMatrix_C[x, length - y].Style.BackColor = SystemColors.ControlLight; }
                            else { dGV_KeyMatrix_C[x, length - y].Value = 0;
dGV_KeyMatrix_C[x, length - y].Style.BackColor = SystemColors.Window; }
                            break;
                        default: //Alfa vs Beta
                            if ((MyArr[y] & (len >> x)) != 0) { dGV_KeyMatrix_A_vs_B[x,
length - y].Value = 1; dGV_KeyMatrix_A_vs_B[x, length - y].Style.BackColor =
SystemColors.ControlLight; }
                            else { dGV_KeyMatrix_A_vs_B[x, length - y].Value = 0;
dGV_KeyMatrix_A_vs_B[x, length - y].Style.BackColor = SystemColors.Window; }
                            break;
                    }
                }
                switch (xD)
                {
                    case 'A': dGV_KeyMatrix_A.Rows[y].Height =
dGV_KeyMatrix_A.Columns[y].Width = 20; break;
                    case 'B': dGV_KeyMatrix_B.Rows[y].Height =
dGV_KeyMatrix_B.Columns[y].Width = 20; break;
                    case 'C': dGV_KeyMatrix_C.Rows[y].Height =
dGV_KeyMatrix_C.Columns[y].Width = 20; break;
                    default: dGV_KeyMatrix_A_vs_B.Rows[y].Height =
dGV_KeyMatrix_A_vs_B.Columns[y].Width = 20; break;
                }
            }
        }

        void OB_A(char pw = 'p')
        {   //НП
            string p_KeyMatrixStr = cB_polynomial_KeyMatrix_A_2.SelectedItem.ToString();
//выбор неприводимого полинома
            int p_KeyMatrixInt = Convert.ToInt32(p_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            if (pw == 'p') cB_element_KeyMatrix_A.DataSource =
MyBitA.bitModXor(p_KeyMatrixInt); //запись образующих элементов
            //ОЭ
            string w_KeyMatrixStr = cB_element_KeyMatrix_A.SelectedItem.ToString(); //выбор
неприводимого полинома
            int w_KeyMatrixInt = Convert.ToInt32(w_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            //Матрица
            tabl_KeyMatrix(p_KeyMatrixInt, w_KeyMatrixInt, 'A', 8);
```

```
        }

        void OB_B(char pw = 'p')
        {   //НП
            string p_KeyMatrixStr = cB_polynomial_KeyMatrix_B_2.SelectedItem.ToString();
//выбор неприводимого полинома
            int p_KeyMatrixInt = Convert.ToInt32(p_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            if (pw == 'p') cB_element_KeyMatrix_B.DataSource =
MyBitA.bitModXor(p_KeyMatrixInt); //запись образующих элементов
            //ОЭ
            string w_KeyMatrixStr = cB_element_KeyMatrix_B.SelectedItem.ToString(); //выбор
неприводимого полинома
            int w_KeyMatrixInt = Convert.ToInt32(w_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            //Матрица
            tabl_KeyMatrix(p_KeyMatrixInt, w_KeyMatrixInt, 'B', 8);
        }

        void OB_C(char pw = 'p')
        {   //НП
            string p_KeyMatrixStr = cB_polynomial_KeyMatrix_C_2.SelectedItem.ToString();
//выбор неприводимого полинома
            int p_KeyMatrixInt = Convert.ToInt32(p_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            if (pw == 'p') cB_element_KeyMatrix_C.DataSource =
MyBitA.bitModXor(p_KeyMatrixInt); //запись образующих элементов
            //ОЭ
            string w_KeyMatrixStr = cB_element_KeyMatrix_C.SelectedItem.ToString(); //выбор
неприводимого полинома
            int w_KeyMatrixInt = Convert.ToInt32(w_KeyMatrixStr, 2); //преобразование строки
с записаным двоичным числом в десятичное число
            //Матрица
            tabl_KeyMatrix(p_KeyMatrixInt, w_KeyMatrixInt, 'C', 8);
        }

        private void linkLabel1_Click(object sender, EventArgs e)
        {
            Form_alg fs = new Form_alg();
            fs.Show();
        }

        private void Label_gisto_Click(object sender, EventArgs e)
        {
            Clipboard.SetText(Label_gisto_in.Text);

            string message = "Вы поместили текстовое значение\n" +
                             Label_gisto_in.Text +
                             "\nВ буфер обмена Windows\n" +
                             "Теперь Вы можете «вставить» его в любой другой
программе.\n" +
                             "Нажав(Ctrl + V)";
            string caption = "Теперь значение энтропии в буфере обмена";

            MessageBox.Show(message, caption, MessageBoxButtons.OK,
MessageBoxIcon.Asterisk);
        }

        private void Label_gisto_out_Click(object sender, EventArgs e)
        {
            Clipboard.SetText(Label_gisto_out.Text);

            string message = "Вы поместили текстовое значение\n" +
```

```
                                Label_gisto_out.Text +
                                "\nВ буфер обмена Windows\n" +
                                "Теперь Вы можете «вставить» его в любой другой
программе.\n" +
                                "Нажав(Ctrl + V)";
            string caption = "Теперь значение энтропии в буфере обмена";

            MessageBox.Show(message, caption, MessageBoxButtons.OK,
MessageBoxIcon.Asterisk);
        }

        private string Test_text(TextBox textBox)
        {
            int round = textBox_Key.Lines.Length;
            string[] keyArray = textBox_Key.Lines;
            string keyString;
            for (int i_round = 0; i_round < round; i_round++)
            {
                keyString = keyArray[i_round];
                if (keyString.Length < 16)
                {
                    textBox.Enabled = true;
                    textBox.ReadOnly = false;
                    textBox.Focus();

                    textBox.Select(18 * (i_round), keyString.Length);

                    MessageBox.Show("Проверьте ключ\n(скорее всего в нем меньше 16 знаков)",
"Неверный ключ", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    return "Error";
                }
                if (keyString.Length > 16)
                   {
                         textBox.Enabled = true;
                    textBox.ReadOnly = false;
                    textBox.Focus();

                    textBox.Select(18 * (i_round), keyString.Length);

                    MessageBox.Show("Проверьте ключ\n(скорее всего в нем больше 16 знаков)",
"Неверный ключ", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    return "Error";
                   }
                if (keyString.Length == 16)
                   {
                    string value;
                    int num;
                    for (byte i = 0; i < 16; i++)
                            {
                        value = keyString[i].ToString();
                        try
                        {
                            num = Convert.ToInt32(value, 16);
                        }
                        catch (Exception)
                        {
                            textBox.Enabled = true;
                            textBox.ReadOnly = false;
                            textBox.Focus();

                            textBox.Select(i + (18 * i_round), 1);
                            MessageBox.Show("Проверьте ключ\n(скорее всего в нем " + value +
" не 16-е число)", "Неверный ключ", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

```
                            return "Error";
                        }
                            }
                    }
                }
                return "OK";
            }

            private void button_Save_KeyMatrix_A_vs_B_Click(object sender, EventArgs e)
            {
                MyStream.File_Save_A(6);
            }

            private void dGV_KeyMatrix_A_vs_B_CellValidating(object sender,
DataGridViewCellValidatingEventArgs e)
            {
                testing_0_or_1(e);
            }

            private void testing_0_or_1(DataGridViewCellValidatingEventArgs e)
            {
                string cellValue = e.FormattedValue.ToString();
                if (cellValue != "1" && cellValue != "0")
                {
                    MessageBox.Show("Значение может быть только 0 или 1");
                    e.Cancel = true;
                }
            }

            private void linkLabel2_Click(object sender, EventArgs e)
            {
                Clipboard.SetText("sg6336@yandex.ru");
                MessageBox.Show(@"Навроцкий Денис
(аспирант)
моб. 050-441-81-35
e-mail: sg6336@yandex.ru

Запись в буфер обмена:
sg6336@yandex.ru

Теперь можно ''Вставить'' из буфера нажатием Ctrl + V",
                "Запись в буфер обмена", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            //SlideCode
            private void rB_B_CheckedChanged(object sender, EventArgs e)
            {
                if (rB_B.Checked) comboBox_block.Enabled = true;
            }

            private void rB_A_CheckedChanged(object sender, EventArgs e)
            {
                if (rB_A.Checked) { comboBox_block.Enabled = false; comboBox_block.SelectedIndex
= 0; }
            }
            //Sub
            private void rB_Sub_XYZ_CheckedChanged(object sender, EventArgs e)
            {
                if (rB_Sub_XYZ.Checked) { cB_Sub_XYZ.Visible = true; cB_Sub_123.Visible = false;
}
            }

            private void rB_Sub_123_CheckedChanged(object sender, EventArgs e)
```

```
        {
            if (rB_Sub_123.Checked) { cB_Sub_XYZ.Visible = false; cB_Sub_123.Visible = true;
}
        }
        //Permut
        private void rB_Perm_Sub_CheckedChanged(object sender, EventArgs e)
        {
            if (rB_Perm_Sub.Checked) { cB_Perm_Sub.Visible = false; cB_Perm_XYZ.Visible =
false; cB_Perm_123.Visible = false; }
        }

        private void rB_Perm_XYZ_CheckedChanged(object sender, EventArgs e)
        {
            if (rB_Perm_XYZ.Checked) { cB_Perm_Sub.Visible = false; cB_Perm_XYZ.Visible =
true; cB_Perm_123.Visible = false; }
        }

        private void rB_Perm_123_CheckedChanged(object sender, EventArgs e)
        {
            if (rB_Perm_123.Checked) { cB_Perm_Sub.Visible = false; cB_Perm_XYZ.Visible =
false; cB_Perm_123.Visible = true; }
        }



    }
}
```

Програмний код класу “GISTO.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Collections; //для BitArray и Hashtable

namespace RSB_64_3D
{
    static class GISTO
    {
        const int kMax = 8; //размер кубика в байтах

        static public double start(byte[] MyArr1)
        {
            int[] MyArr2 = new int[256];
            int N; //количество байт в файле
            double H; //энтропия

            Buffer.BlockCopy(GISTO_e(MyArr1), 0, MyArr2, 0, 0x400); //получаем массив байт
ГИСТО (указано сколько каких байт) &H400 - 256*4=1024 байта, потому что UInteger это 4 байта
            N = MyArr1.Length;
            H = GISTO_entropy(MyArr2, N);

            return H;
        }

        static int[] GISTO_e(byte[] MyArr1)
        {
            int[] myArr2 = new int[256];
            int value;
```

```
    //byte index;

    //Значение из массива MyArr1 делаю индексом массива MyArr2.
    //и полученное значение по этому индексу увеличиваю на 1
    foreach (byte index in MyArr1)
    {
        value = myArr2[index];
        value += 1;
        myArr2[index] = value;
    }
    return myArr2;
}

static double GISTO_entropy(int[] MyArr2, int N)
{
    double p, H=0;

    foreach (int value in MyArr2)
    {
        p = (double) ((double)value / (double)N);
        if (p == 0.0) continue;
        p *= (-Math.Log(p,2));
        H += p;
    }
    return H;
}

public static string[] MyArrKeyF(string[] MyArrKey, int step)
{
    string strKey;

    int round = MyArrKey.Length;
    BitArray myList;
    bool[, ,] MyArray3D_new = new bool[4, 4, 4];
    int j, i, k = 0, x, y, z;
    byte[] randomKeyNew = new byte[kMax * round]; //полный ключ в байтах
    byte[] MyArr1D = new byte[kMax]; //выходной массив байт для одной строки
    BitArray MyArrBool = new BitArray(64);

    string[] MyStrKey = new string[round];
    string[] Text_KeyF = new string[round];

    bool flagBreak = false; //для завершения цикла

    for (int i_round = 0; i_round < round; i_round++)
    {
        flagBreak = false;
        strKey = MyArrKey[i_round];
        //преобразовываем строку из массива в массив байт
        for (j = 0, i = 0; i < strKey.Length; i += 2)
        {
            randomKeyNew[j++] = Convert.ToByte(strKey.Substring(i, 2), 16);
        }
        //преобразовываем массив байт в массив бит
        myList = new BitArray(randomKeyNew);

        //преобразовываем массив бит в трехмерный логический массив
        //Создадим НОВЫЙ 3-D массив bit
        // Для шагов 1,2,3,4,5,6
        for (x = 0; x < 4; x++)
        {
            for (y = 0; y < 4; y++)
            {
```

```
            for (z = 0; z < 4; z++)
            {
                switch (step)
                {
                    case 0:
                        MyArray3D_new[x, y, z] = myList[k++];
                        break;
                    case 1:
                        MyArray3D_new[y, z, x] = myList[k++];
                        break;
                    case 2:
                        MyArray3D_new[z, x, y] = myList[k++];
                        break;
                    case 3:
                        MyArray3D_new[y, x, z] = myList[k++];
                        break;
                    case 4:
                        MyArray3D_new[x, z, y] = myList[k++];
                        break;
                    case 5:
                        MyArray3D_new[z, y, x] = myList[k++];
                        break;
                    default:
                        flagBreak = true;
                        break;
                }
                if (flagBreak) break;
            }
            if (flagBreak) break;
        }
        if (flagBreak) break;
    }
    flagBreak = false;

    // Для шагов 7,8,9,10,11,12
    for (x = 3; x >= 0 ; x--)
    {
        for (y = 0; y < 4; y++)
        {
            for (z = 0; z < 4; z++)
            {
                switch (step)
                {
                    case 6:
                        MyArray3D_new[x, y, z] = myList[k++];
                        break;
                    case 7:
                        MyArray3D_new[y, z, x] = myList[k++];
                        break;
                    case 8:
                        MyArray3D_new[z, x, y] = myList[k++];
                        break;
                    case 9:
                        MyArray3D_new[y, x, z] = myList[k++];
                        break;
                    case 10:
                        MyArray3D_new[x, z, y] = myList[k++];
                        break;
                    case 11:
                        MyArray3D_new[z, y, x] = myList[k++];
                        break;
                    default:
                        flagBreak = true;
```

```
                    break;
                }
                if (flagBreak) break;
            }
            if (flagBreak) break;
        }
        if (flagBreak) break;
    }
    flagBreak = false;

    // Для шагов 13,14,15,16,17,18
    for (x = 3; x >= 0; x--)
    {
        for (y = 3; y >= 0; y--)
        {
            for (z = 0; z < 4; z++)
            {
                switch (step)
                {
                    case 12:
                        MyArray3D_new[x, y, z] = myList[k++];
                        break;
                    case 13:
                        MyArray3D_new[y, z, x] = myList[k++];
                        break;
                    case 14:
                        MyArray3D_new[z, x, y] = myList[k++];
                        break;
                    case 15:
                        MyArray3D_new[y, x, z] = myList[k++];
                        break;
                    case 16:
                        MyArray3D_new[x, z, y] = myList[k++];
                        break;
                    case 17:
                        MyArray3D_new[z, y, x] = myList[k++];
                        break;
                    default:
                        flagBreak = true;
                        break;
                }
                if (flagBreak) break;
            }
            if (flagBreak) break;
        }
        if (flagBreak) break;
    }
    flagBreak = false;

    // Для шагов 19,20,21,22,23,24
    for (x = 3; x >= 0; x--)
    {
        for (y = 3; y >= 0; y--)
        {
            for (z = 3; z >= 0; z--)
            {
                switch (step)
                {
                    case 18:
                        MyArray3D_new[x, y, z] = myList[k++];
                        break;
                    case 19:
                        MyArray3D_new[y, z, x] = myList[k++];
```

```
                                    break;
                                case 20:
                                    MyArray3D_new[z, x, y] = myList[k++];
                                    break;
                                case 21:
                                    MyArray3D_new[y, x, z] = myList[k++];
                                    break;
                                case 22:
                                    MyArray3D_new[x, z, y] = myList[k++];
                                    break;
                                case 23:
                                    MyArray3D_new[z, y, x] = myList[k++];
                                    break;
                                default:
                                    flagBreak = true;
                                    break;
                            }
                            if (flagBreak) break;
                        }
                        if (flagBreak) break;
                    }
                    if (flagBreak) break;
                }
                flagBreak = false;

                //Перейдем от массива 3-D к одномерному массиву
                k = 0;
                for (x = 0; x < 4; x++)
                {
                    for (y = 0; y < 4; y++)
                    {
                        for (z = 0; z < 4; z++)
                        {
                            MyArrBool[k++] = MyArray3D_new[x, y, z];
                        }
                    }
                }

                //преобразование битов в байты
                MyArrBool.CopyTo(MyArr1D, 0);

                //только первые четыре байта масива копируем. Из них выйдет 8 символов
                MyStrKey = MyArr1D.Select(num => Convert.ToString(num, 16).PadLeft(2,
'0')).ToArray(); //Если строка из двух символов не получается, То добавляем 0

                for (k = 0; k < kMax; k++) //берем весь ключ!!!!!!!!!!!!!!!!!!
                {
                    Text_KeyF[i_round] += MyStrKey[k].ToUpper(); //для ключа, беререм первые
16-ть символов (8-мь двойных символа)
                }
                k = 0;
            }

            return Text_KeyF;
        }
    }
}
```

Програмний код класу “MyBitA.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;

using System.IO;
using System.Collections; //для BitArray и Hashtable
using System.Security.Cryptography; //для RandomNumberGenerator.GetBytes - метод

namespace RSB_64_3D
{
    static class MyBitA
    {
        const int kMax = 8; //размер кубика в байтах
        public static byte[][] KeyMatrix2D;
        public static byte key_Alfa = 0, key_Beta = 0; //для Permut-Sub
        //public static Form_MO f_MO;

        //Заполняем контрол "Образующий элемент"
        static public string[] bitModXor(int p, int items = kMax)
        {
            int bits, Lmax, i_Lmax, degree;

            if (p <= 0x80)
            {
                bits = 0x3F;
                items = 6;
            }
            else bits = 0xFF; //=0x3F;
            Lmax = (int)(bits / 3);

            degree = (int)(Math.Pow(2, items) - 1);

            int w, www, var_w, p_old;
            int i_round, flag, i, round;
            int Ps;

            int[] MyArr3 = new int[bits - 2]; //выходные данные (для построения таблицы)
            int k = 0;

            int[] array_w = new int[items];
            int var_array_w;
            int DeciValue, NoOfBits;

            //int kkk = 0;
            //bool CheckB1 = true;

            for (www = 2; www <= degree; www++)
            {
                Array.Clear(array_w, 0, items); //обнуление массива
                DeciValue = www;
                NoOfBits = 1;

                while (DeciValue > Math.Pow(2, NoOfBits) - 1)
                {
                    NoOfBits = NoOfBits + 1;
                }

                for (i = 0; i < items; i++)
                {
                    array_w.SetValue(Convert.ToInt32((DeciValue &
Convert.ToInt32(Math.Pow(2, i))) / Math.Pow(2, i)), i);
                }

                round = NoOfBits - 1;
```

```
                i = 0;
                w = www;
                Ps = 1;
                var_w = 0;
                //Умножение стольбиком, а потом сложение по модулю 2
                for (i_Lmax = 0; i_Lmax <= Lmax; i_Lmax++)
                {
                    Ps++;
                    var_w = 0;
                    //умножение w << i_round и сложение по модулю var_w Xor (w << i_round)
                    for (i_round = 0; i_round <= round; i_round++)
                    {
                        var_array_w = Convert.ToInt32(array_w.GetValue(i_round));
                        if (var_array_w == 1)
                        {
                            var_w = var_w ^ (w << i_round);
                        }
                    }
                    //Если вылазит за лишний разряд, то:
                    while (var_w > bits)
                    {
                        p_old = p;
                        flag = var_w >> items;
                        if (flag > 1) //Если перебор больше чем 1 разряд, то начинаем
сдвигать полином
                        {
                            i = 0;
                            while (flag != 1)
                            {
                                flag = flag >> 1;
                                i++;
                            }
                            p = p << i;
                        }
                        var_w = var_w ^ p;
                        p = p_old;
                    }
                    w = var_w;
                    if (var_w == 1)
                    {
                        break;
                    }
                }
                if (Ps == bits || (i_Lmax - 1) == Lmax & var_w != 1)
                {
                    MyArr3.SetValue(www, k);
                    k += 1;
                }
            }

            Array.Resize(ref MyArr3, k);

            string[] MyArr = MyArr3.Select(num => Convert.ToString(num, 2).PadLeft(items,
'0')).ToArray(); //преобразовую число в строку (двоичное число)

            return MyArr;
        }

        //sub XYZ 3D, подготовка таблиц MO, SBox, Reverse_SBox
        static public void Tabl(int w, int p, ref byte[,] MyMO_2D, ref byte[,]
MyArrSBox_2D_A, ref byte[,] MyArrReverseSBox_2D_A, bool bildTabl, char xD = '0')
        {
```

```
            bool[] array_w = Convert.ToString(w, 2).Select(num => Convert.Equals(num,
'1')).ToArray(); //преобразовую число в строку (двоичное число), а потом в битовый массив
True и False
            Array.Reverse(array_w);

            int items = kMax;
            int bits;
            if (p <= 0x80)
            {
                bits = 0x3F;
                items = 6;
            }
            else { bits = 0xFF; items = 8; }

            int[] MyArr2 = new int[bits + 1];
            MyArr2[0] = 1;
            MyArr2[1] = w;
            int var_w = 0;
            int p_old, i_round, flag, i;
            int Ps = 1;
            int round = array_w.GetUpperBound(0);

            //Умножение стольбиком, а потом сложение по модулю 2
            while (var_w != 1)
            {
                Ps++;
                var_w = 0;
                //умножение w << i_round и сложение по модулю var_w Xor (w << i_round)
                for (i_round = 0; i_round <= round; i_round++)
                {
                    if (array_w[i_round] == true)
                    {
                        var_w ^= (w << i_round);
                    }
                }
                //Если вылазит за лишний разряд, то:
                while (var_w > bits)
                {
                    p_old = p;
                    flag = var_w >> items;
                    if (flag > 1) //Если перебор больше чем 1 разряд, то начинаем сдвигать
полином
                    {
                        i = 0;
                        while (flag != 1)
                        {
                            flag >>= 1;
                            i++;
                        }
                        p <<= i;
                    }
                    var_w ^= p;
                    p = p_old;
                }
                w = var_w;
                MyArr2[Ps] = w;
            }

            int bitModXor = Ps;

            byte[] MyArrMO = new byte[Ps + 1];
            byte[] MyArrSBox = new byte[Ps + 1];
            byte[] MyArrReverseSBox = new byte[Ps + 1];
```

```
            if (bitModXor == bits)
            {
                Array.Resize(ref MyArr2, Ps + 1);
                Buffer.BlockCopy(Tabl_MO(MyArr2, bits), 0, MyArrMO, 0, Ps + 1); //вычисляем
массив данных для таблицы MO
                Buffer.BlockCopy(Tabl_S_Box_Alfa_vs_Beta(MyArrMO, bits, items), 0,
MyArrSBox, 0, Ps + 1); //вычисляем массив данных для таблицы SBox

                //записуем в двумерные массивы для шифрования //для "выхода из функции"
                int arX, arY;
                i = 0;
                int len;
                if (items == kMax) len = 0xF;
                else len = 0x7;
                for (arX = 0; arX <= len; arX++)
                {
                    for (arY = 0; arY <= len; arY++)
                    {
                        MyMO_2D[arX, arY] = MyArrMO[i];
                        MyArrSBox_2D_A[arX, arY] = MyArrSBox[i++];
                    }
                }
                Buffer.BlockCopy(Tabl_Reverse_S_Box_Alfa_vs_Beta(MyArrSBox_2D_A), 0,
MyArrReverseSBox_2D_A, 0, Ps + 1); //вычисляем массив данных для таблицы обратной SBox
            }
            ////строим таблицу
            //if (bildTabl) //Если есть данные для таблицы, то:
            //{
            //    //Form_MO f_MO = new Form_MO();
            //    if (f_MO == null)
            //    {
            //        f_MO = new Form_MO();
            //    }
            //    f_MO.xD = xD;
            //    f_MO.DataGrid_MO(MyArrMO);
            //    f_MO.DataGrid_S_Box(MyArrSBox_2D_A);
            //    f_MO.DataGrid_Reverse_S_Box(MyArrReverseSBox_2D_A);
            //    f_MO.Show();
            //}
        }

        public static byte[] Tabl_MO(int[] MyArr0, int bits)
        {
            int MyArr0_length = MyArr0.Length;
            int[] MyArr1 = new int[MyArr0_length];
            int[] MyArr2 = new int[MyArr0_length];
            byte[] arr_MO = new byte[bits + 1];
            Buffer.BlockCopy(MyArr0, 0, MyArr1, 0, MyArr0_length * 4); //MyArr1 - массив
степеней
            Buffer.BlockCopy(MyArr0, 0, MyArr2, 0, MyArr0_length * 4);
            Array.Reverse(MyArr2); //MyArr2 - массив степеней с инверсией
            //сдвигаем весь массив на одну позцию (с потерей последней строки)
            //в первую строку пишем 0
            int i;
            //int val;
            for (i = MyArr0_length - 2; i >= 0; i--)
            {
                MyArr1[i + 1] = MyArr1[i];
                MyArr2[i + 1] = MyArr2[i];
            }
            MyArr1[0] = MyArr2[0] = 0;
```

```
            //"Ранжирование". Массив3 = значение массива 1 по индексу массива 2.
            //Подряд
            for (i = 0; i < MyArr0_length; i++)
            {
                arr_MO[MyArr1[i]] = (byte)MyArr2[i];
            }
            //Передаем значения в форму
            return arr_MO;
        }

        public static byte[] Tabl_S_Box_Alfa_vs_Beta(byte[] MyArrMO, int bits, int kkMax)
        {
            byte MO_and_KeyMatrix2D; //= new int[kMax]; //умножение MO на A (i_colum
столбец)
            int sum, i_colum, i = 0;
            byte[] MyArrMO_val = new byte[MyArrMO.Length];
            Buffer.BlockCopy(MyArrMO, 0, MyArrMO_val, 0, MyArrMO.Length);

            byte[] ArrkkMax = new byte[1] { key_Beta };
            BitArray BitArrBeta = new BitArray(ArrkkMax); //Бета

            BitArray BitArr = new BitArray(8); //kkMax);
            byte[] s_boxT = new byte[bits + 1]; //финальный массив

            for (int i_bit = 0; i_bit <= bits; i_bit++)
            {
                if (key_Alfa != 0) MyArrMO_val[i_bit] ^= key_Alfa;
                //Битовое представление MO
                for (i_colum = 0; i_colum < kkMax; i_colum++)
                {
                    MO_and_KeyMatrix2D = (byte)(MyArrMO_val[i_bit] &
KeyMatrix2D[i_colum][0]); //побитово умножаем
                    sum = 0; //обнуляем
                    //Находим сумму 1 в массиве
                    for (i = 0; i < kkMax; i++) { if ((MO_and_KeyMatrix2D & (1 << i)) != 0)
{ sum++; } }
                    BitArr[kkMax - 1 - i_colum] = Convert.ToBoolean(sum % 2);
                }
                if (key_Beta != 0) BitArr.Xor(BitArrBeta);

                BitArr.CopyTo(s_boxT, i_bit);
            }
            return s_boxT;
        }

        public static byte[,] Tabl_Reverse_S_Box_Alfa_vs_Beta(byte[,] MyArr2D_SBox)
        {
            //находим обратный SBox
            int len = (int)Math.Sqrt(MyArr2D_SBox.Length) - 1;
            byte[,] MyArr2D_ReverseSBox = new byte[len + 1, len + 1];
            byte columnX, rowY, val, valL, valH;
            //это 2D массив ReverseSBox (обратный SBox)
            for (columnX = 0; columnX <= len; columnX++)
            {
                for (rowY = 0; rowY <= len; rowY++)
                {
                    val = MyArr2D_SBox[columnX, rowY];
                    if (len == 7)
                    {
                        valH = (byte)(val >> 3);
                        valL = (byte)(val - (valH << 3));
                        MyArr2D_ReverseSBox[valH, valL] = (byte)((columnX << 3) + rowY);
                    }
```

```
                    else
                    {
                        valH = (byte)(val >> 4);
                        valL = (byte)(((byte)(val << 4)) >> 4); //(byte) (val - (valH << 
4));
                        MyArr2D_ReverseSBox[valH, valL] = (byte)((columnX << 4) + rowY);
                    }
                }
            }
            return MyArr2D_ReverseSBox;
        }
    }
}
```

Програмний код класу “MyCrypt.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Collections; //для BitArray и Hashtable

namespace RSB_64_3D
{
    class MyCrypt
    {
        const int kMax = 8; //размер кубика в байтах

        public static void crypt_SlaideCode_in(ref List<bool[, ,]> listFile, int 
i_round, bool[, ,] arrKeyRound, int blokCub, bool flag_A) //возможно нужно будет 
добавиь return List
        {
            //Пишем код для Слайд Кода
            SlaideCode.flagCript = true; //включзачем зашифровку данных
            SlaideCode.SlaideCode_crypt(ref listFile, i_round, arrKeyRound, blokCub, 
flag_A);
        }

        public static void crypt_SlaideCode_out(ref List<bool[, ,]> listFile, int 
i_round, bool[, ,] arrKeyRound, int blokCub, bool flag_A) //возможно нужно будет 
добавиь return List
        {
            //Пишем код для Слайд Кода
            SlaideCode.flagCript = false; //включзачем расшифровку данных
            SlaideCode.SlaideCode_crypt(ref listFile, i_round, arrKeyRound, blokCub, 
flag_A);
        }

        public static void crypt_Sub_in(ref List<bool[, ,]> listFile, byte[,] 
MyArrSBox_2D_A, byte uArr) //возможно нужно будет добавиь return List
        {
            //Пишем код sub_XYZ_3D
            SubCode_3D.SubCode_3D_cript(ref listFile, MyArrSBox_2D_A, uArr);
        }

        public static void crypt_Sub_out(ref List<bool[, ,]> listFile, byte[,] 
MyArrReverseSBox_2D_A, byte uArr) //возможно нужно будет добавиь return List
        {
            //Пишем код sub_XYZ_3D
            SubCode_3D.SubCode_3D_cript(ref listFile, MyArrReverseSBox_2D_A, uArr);
```

```
        }

        public static void crypt_Permut_SUB_in(ref List<bool[, ,]> listFile, byte[,]
MyArrSBox_2D_A, byte uArr) //возможно нужно будет добавиь return List
        {
            //Пишем код Permut Sub α vs β_3D
            Permut_SUB_3D.Permut_SUB_3D_cript(ref listFile, MyArrSBox_2D_A, uArr);
        }

        public static void crypt_Permut_SUB_out(ref List<bool[, ,]> listFile, byte[,]
MyArrReverseSBox_2D_A, byte uArr) //возможно нужно будет добавиь return List
        {
            //Пишем код Permut Sub α vs β_3D
            Permut_SUB_3D.Permut_SUB_3D_cript(ref listFile, MyArrReverseSBox_2D_A,
uArr);
        }

        public static void crypt_Permut_XYZ_in(ref List<bool[, ,]> listFile, string
textX, string textY, string textZ, byte Perm_XYZ) //возможно нужно будет добавиь return
List
        {
            //Пишем код Permut_3D
            Permut_XYZ.flagCript = true;
            Permut_XYZ.Permut_XYZ_3D_cript(ref listFile, textX, textY, textZ,
Perm_XYZ);
        }

        public static void crypt_Permut_XYZ_out(ref List<bool[, ,]> listFile, string
textX, string textY, string textZ, byte Perm_XYZ) //возможно нужно будет добавиь return
List
        {
            //Пишем код Permut_3D
            Permut_XYZ.flagCript = false;
            Permut_XYZ.Permut_XYZ_3D_cript(ref listFile, textX, textY, textZ,
Perm_XYZ);
        }
    }
}
```

Програмний код класу “ MyStream.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.IO;
using System.Windows.Forms;

namespace RSB_64_3D
{
    static class MyStream
    {
        public static int Filter;
        public static string strFileOpen = "";
        public static string strFileSave = "";

        public static byte[] File_in
        {
            get
```

```
            {
                Stream myStream = null;
                OpenFileDialog openFileDialog1 = new OpenFileDialog();

                openFileDialog1.InitialDirectory = "c:\\";
                openFileDialog1.Filter = "Все файлы (*.*)|*.*|Файлы cript
(*.cript)|*.cript|Документы Word (*.doc;*.rtf;*.docx)|*.doc;*.rtf;*.docx|Текстовые
файлы (*.txt)|*.txt";

                openFileDialog1.FilterIndex = Filter;
                openFileDialog1.RestoreDirectory = true;

                if (openFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    try
                    {
                        if ((myStream = openFileDialog1.OpenFile()) != null)
                        {
                            strFileOpen = openFileDialog1.FileName;
                            //Чтение данных из файла и запись их в одномерный массив
байт.
                            int numBytesToRead = (int)myStream.Length;
                            byte[] bytes = new byte[numBytesToRead];
                            myStream.Read(bytes, 0, numBytesToRead);

                            return bytes;
                        }
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show("Error: Could not read file from disk. Original
error: " + ex.Message);
                    }
                }
                return null;
            }
        }

        public static Stream File_out
        {
            get
            {
                Stream myStream;
                SaveFileDialog saveFileDialog1 = new SaveFileDialog();

                saveFileDialog1.Filter = "Все файлы (*.*)|*.*|Файлы cript
(*.cript)|*.cript|Документы Word (*.doc;*.rtf;*.docx)|*.doc;*.rtf;*.docx|Текстовые
файлы (*.txt)|*.txt";
                saveFileDialog1.FilterIndex = Filter;
                saveFileDialog1.RestoreDirectory = true;

                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    if ((myStream = saveFileDialog1.OpenFile()) != null)
                    {
                        strFileSave = saveFileDialog1.FileName;
                        // Code to write the stream goes here.
                        return myStream;
                        //myStream.Close();
```

```
                    }
                }
                return null;
            }
        }

        public static Stream File_Save_A(int round)
        {
                Stream myStream;
                SaveFileDialog saveFileDialog1 = new SaveFileDialog();

                saveFileDialog1.Filter = "Все файлы (*.*)|*.*|Файлы cript
(*.cript)|*.cript|Документы Word (*.doc;*.rtf;*.docx)|*.doc;*.rtf;*.docx|Текстовые
файлы (*.txt)|*.txt";
                saveFileDialog1.FilterIndex = Filter;
                saveFileDialog1.RestoreDirectory = true;

                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    if ((myStream = saveFileDialog1.OpenFile()) != null)
                    {
                        strFileSave = saveFileDialog1.FileName;
                        // Code to write the stream goes here.
                        return myStream;
                        //myStream.Close();

        //                Dim sw As System.IO.StreamWriter = New
System.IO.StreamWriter(strFileSaveA)

        //Dim keyA(round - 1) As Byte

        //For y As Byte = 0 To 5 Step 1
        //    For x As Byte = 0 To 5 Step 1
        //        keyA.SetValue(CByte(Form1.DataGridViewA.Item(x, y).Value), x)
        //    Next
        //    sw.WriteLine(keyA(0) & keyA(1) & keyA(2) & keyA(3) & keyA(4) & keyA(5))
        //Next

        //sw.Close()
                    }
                }
                return null;

        }
    }
}
```

Програмний код класу “Permut_SUB_3D.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Collections; //для BitArray и Hashtable

namespace RSB_64_3D
{
    static class Permut_SUB_3D
    {
```

```
        const int kMax = 8; //размер кубика в байтах

        public static void Permut_SUB_3D_cript(ref List<bool[, ,]> listFile, byte[,]
MyArrSBox_2D_A, byte uArr)
        {
            BitArray value_bit; //64 бита
            BitArray value_bit_new = new BitArray(64);
            int i, k;
            //для 1-D преобразования
            byte vL; //младший полубайт
            byte vH; //старший полубайт
            byte v; //байт

            byte v_SBox; //байт из таблицы S-Box
            bool v_bit;

            //начинаем саму обработку кубиков
            int iMax = listFile.Count; //количество кубиков в файле
            for (i = 0; i < iMax; i++)
            {
                value_bit = Array_3D_to_1D(listFile[i], uArr); //получаем 1D байтовый
массив (из кубика)

                k = 0;
                for (vH = 0; vH < kMax; vH++)
                {
                    for (vL = 0; vL < kMax; vL++)
                    {
                        v_SBox = MyArrSBox_2D_A[vH, vL];
                        v_bit = value_bit[v_SBox];
                        value_bit_new[k++] = v_bit;
                    }
                }
                //преобразовываем 1D BitArray в 3D bool[,,] и записываем в массив файла
                listFile[i] = Array_1D_to_3D(value_bit_new, uArr);
            }
        }

        public static BitArray Array_3D_to_1D(bool[, ,] value64, byte uArr)
        {
            //Делаем выборку из 3D массива, согласно схеме кубиков. И собираем 1D
массив
            //byte[] MyArr1 = new byte[kMax];
            BitArray MyArr2 = new BitArray(64);
            //bool[] MyArr2 = new bool[64];
            int i = 0, z, y, x;
            switch (uArr)
            {
                case 1:
                    for (z = 0; z < 4; z++)
                    {
                        for (y = 0; y < 4; y++)
                        {
                            for (x = 0; x < 4; x++)
                            {
                                MyArr2[i++] = value64[z, y, x];
                            }
                        }
                    }
```

```
                break;
            case 2:
                for (x = 0; x < 4; x++)
                {
                    for (z = 0; z < 4; z++)
                    {
                        for (y = 0; y < 4; y++)
                        {
                            MyArr2[i++] = value64[z, y, x];
                        }
                    }
                }
                break;
            case 3:
                for (y = 0; y < 4; y++)
                {
                    for (x = 0; x < 4; x++)
                    {
                        for (z = 0; z < 4; z++)
                        {
                            MyArr2[i++] = value64[z, y, x];
                        }
                    }
                }
                break;
            default:
                break;
        }

        //преобразование битов в байты
        //MyArr2.CopyTo(MyArr1, 0);

        return MyArr2; //одномерный массив байт
    }

    public static bool[, ,] Array_1D_to_3D(BitArray myList, byte uArr)
    {
        int i = 0, z, y, x;
        bool[, ,] MyArray3D = new bool[4, 4, 4];

        //Преобразуем массив byte в bit
        //BitArray myList = new BitArray(value);
        switch (uArr)
        {
            case 1:
                for (z = 0; z < 4; z++)
                {
                    for (y = 0; y < 4; y++)
                    {
                        for (x = 0; x < 4; x++)
                        {
                            MyArray3D[z, y, x] = myList[i++];
                        }
                    }
                }
                break;
            case 2:
                for (x = 0; x < 4; x++)
                {
```

```
                        for (z = 0; z < 4; z++)
                        {
                            for (y = 0; y < 4; y++)
                            {
                                MyArray3D[z, y, x] = myList[i++];
                            }
                        }
                    }
                    break;
                case 3:
                    for (y = 0; y < 4; y++)
                    {
                        for (x = 0; x < 4; x++)
                        {
                            for (z = 0; z < 4; z++)
                            {
                                MyArray3D[z, y, x] = myList[i++];
                            }
                        }
                    }
                    break;
                default:
                    break;
            }
            return MyArray3D; //трехмерный массив bool
        }
    }
}
```

Програмний код класу “Permut_XYZ.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Collections;

namespace RSB_64_3D
{
    static class Permut_XYZ
    {
        const int kMax = 8; //размер кубика в байтах
        static byte[] permutationArrayX = new byte[2 * kMax]; //массив сдвигов для X
        static byte[] permutationArrayY = new byte[2 * kMax]; //массив сдвигов для Y
        static byte[] permutationArrayZ = new byte[2 * kMax]; //массив сдвигов для Z
        static byte varXYZ;
        static public bool flagCript;

        public static void Permut_XYZ_3D_cript(ref List<bool[, ,]> listFile, string
textX, string textY, string textZ, byte Perm_XYZ)
        {
            for (int i = 0, x = 0, y = 0, z = 0; i < kMax; i++)
            {
                varXYZ = Convert.ToByte(textX.Substring(i, 1), 16);
                permutationArrayX[x++] = (byte)(varXYZ >> 2);
                permutationArrayX[x++] = (byte)(varXYZ - (byte)((varXYZ >> 2) << 2));

                varXYZ = Convert.ToByte(textY.Substring(i, 1), 16);
                permutationArrayY[y++] = (byte)(varXYZ >> 2);
```

```
                permutationArrayY[y++] = (byte)(varXYZ - (byte)((varXYZ >> 2) << 2));

                varXYZ = Convert.ToByte(textZ.Substring(i, 1), 16);
                permutationArrayZ[z++] = (byte)(varXYZ >> 2);
                permutationArrayZ[z++] = (byte)(varXYZ - (byte)((varXYZ >> 2) << 2));
            }

            ChangeArray(ref listFile, Perm_XYZ);
        }

        //Количество поворотов по оси x,y,z
        public static void ChangeArray(ref List<bool[, ,]> listFile, byte Perm_XYZ)
        {
            byte ix, iy, iz;

            switch (Perm_XYZ)
            {
                case 1:
                    for (int i = 0; i < listFile.Count; i++)
                    {
                        ix = iy = iz = 0;
                        bool[, ,] MyArray3D = listFile[i]; //???
                        foreach (byte item in permutationArrayX)
                        {
                            if (item != 0) ChangeArray3D_X(ref MyArray3D, item, iy,
iz);

                            iz++;
                            if (iz == 4)
                            {
                                iz = 0;
                                iy++;
                            }
                        }
                    }

                    break;
                case 2:
                    for (int i = 0; i < listFile.Count; i++)
                    {
                        ix = iy = iz = 0;
                        bool[, ,] MyArray3D = listFile[i]; //???
                        foreach (byte item in permutationArrayY)
                        {
                            if (item != 0) ChangeArray3D_Y(ref MyArray3D, item, ix,
iz);

                            ix++;
                            if (ix == 4)
                            {
                                ix = 0;
                                iz++;
                            }
                        }
                    }
                    break;
                case 3:
                    for (int i = 0; i < listFile.Count; i++)
                    {
                        ix = iy = iz = 0;
                        bool[, ,] MyArray3D = listFile[i]; //???
```

```
                        foreach (byte item in permutationArrayZ)
                        {
                            if (item != 0) ChangeArray3D_Z(ref MyArray3D, item, ix,
iy);
                            iy++;
                            if (iy == 4)
                            {
                                iy = 0;
                                ix++;
                            }
                        }
                    }
                    break;
                default:
                    break;
            }
        }

        public static void ChangeArray3D_X(ref bool[, ,] MyArray3D, byte arg, byte y,
byte z)
        {
            bool value_x;

            switch (arg)
            {
                case 1:
                    if (flagCript)
                    {
                        value_x = MyArray3D[1, y, z];

                        MyArray3D[1, y, z] = MyArray3D[0, y, z];
                        MyArray3D[0, y, z] = MyArray3D[3, y, z];
                        MyArray3D[3, y, z] = MyArray3D[2, y, z];
                        MyArray3D[2, y, z] = value_x;
                    }
                    else
                    {
                        value_x = MyArray3D[3, y, z];

                        MyArray3D[3, y, z] = MyArray3D[0, y, z];
                        MyArray3D[0, y, z] = MyArray3D[1, y, z];
                        MyArray3D[1, y, z] = MyArray3D[2, y, z];
                        MyArray3D[2, y, z] = value_x;
                    }
                    break;
                case 2:
                    value_x = MyArray3D[2, y, z];

                    MyArray3D[2, y, z] = MyArray3D[0, y, z];
                    MyArray3D[0, y, z] = value_x;

                    value_x = MyArray3D[3, y, z];

                    MyArray3D[3, y, z] = MyArray3D[1, y, z];
                    MyArray3D[1, y, z] = value_x;
                    break;
                case 3:
                    if (flagCript)
                    {
```

```
                    value_x = MyArray3D[3, y, z];

                    MyArray3D[3, y, z] = MyArray3D[0, y, z];
                    MyArray3D[0, y, z] = MyArray3D[1, y, z];
                    MyArray3D[1, y, z] = MyArray3D[2, y, z];
                    MyArray3D[2, y, z] = value_x;
                }
                else
                {
                    value_x = MyArray3D[1, y, z];

                    MyArray3D[1, y, z] = MyArray3D[0, y, z];
                    MyArray3D[0, y, z] = MyArray3D[3, y, z];
                    MyArray3D[3, y, z] = MyArray3D[2, y, z];
                    MyArray3D[2, y, z] = value_x;
                }
                break;
            default:
                break;
        }
    }

    public static void ChangeArray3D_Y(ref bool[, ,] MyArray3D, byte arg, byte x,
byte z)
    {
        bool value_y;

        switch (arg)
        {
            case 1:
                if (flagCript)
                {
                    value_y = MyArray3D[x, 1, z];

                    MyArray3D[x, 1, z] = MyArray3D[x, 0, z];
                    MyArray3D[x, 0, z] = MyArray3D[x, 3, z];
                    MyArray3D[x, 3, z] = MyArray3D[x, 2, z];
                    MyArray3D[x, 2, z] = value_y;
                }
                else
                {
                    value_y = MyArray3D[x, 3, z];

                    MyArray3D[x, 3, z] = MyArray3D[x, 0, z];
                    MyArray3D[x, 0, z] = MyArray3D[x, 1, z];
                    MyArray3D[x, 1, z] = MyArray3D[x, 2, z];
                    MyArray3D[x, 2, z] = value_y;
                }
                break;
            case 2:
                value_y = MyArray3D[x, 2, z];

                MyArray3D[x, 2, z] = MyArray3D[x, 0, z];
                MyArray3D[x, 0, z] = value_y;

                value_y = MyArray3D[x, 3, z];

                MyArray3D[x, 3, z] = MyArray3D[x, 1, z];
                MyArray3D[x, 1, z] = value_y;
```

```
                    break;
                case 3:
                    if (flagCript)
                    {
                        value_y = MyArray3D[x, 3, z];

                        MyArray3D[x, 3, z] = MyArray3D[x, 0, z];
                        MyArray3D[x, 0, z] = MyArray3D[x, 1, z];
                        MyArray3D[x, 1, z] = MyArray3D[x, 2, z];
                        MyArray3D[x, 2, z] = value_y;
                    }
                    else
                    {
                        value_y = MyArray3D[x, 1, z];

                        MyArray3D[x, 1, z] = MyArray3D[x, 0, z];
                        MyArray3D[x, 0, z] = MyArray3D[x, 3, z];
                        MyArray3D[x, 3, z] = MyArray3D[x, 2, z];
                        MyArray3D[x, 2, z] = value_y;
                    }
                    break;
                default:
                    break;
            }
        }

        public static void ChangeArray3D_Z(ref bool[, ,] MyArray3D, byte arg, byte x,
byte y)
        {
            bool value_z;

            switch (arg)
            {
                case 1:
                    if (flagCript)
                    {
                        value_z = MyArray3D[x, y, 1];

                        MyArray3D[x, y, 1] = MyArray3D[x, y, 0];
                        MyArray3D[x, y, 0] = MyArray3D[x, y, 3];
                        MyArray3D[x, y, 3] = MyArray3D[x, y, 2];
                        MyArray3D[x, y, 2] = value_z;
                    }
                    else
                    {
                        value_z = MyArray3D[x, y, 3];

                        MyArray3D[x, y, 3] = MyArray3D[x, y, 0];
                        MyArray3D[x, y, 0] = MyArray3D[x, y, 1];
                        MyArray3D[x, y, 1] = MyArray3D[x, y, 2];
                        MyArray3D[x, y, 2] = value_z;
                    }
                    break;
                case 2:
                    value_z = MyArray3D[x, y, 2];

                    MyArray3D[x, y, 2] = MyArray3D[x, y, 0];
                    MyArray3D[x, y, 0] = value_z;
```

```
                value_z = MyArray3D[x, y, 3];

                MyArray3D[x, y, 3] = MyArray3D[x, y, 1];
                MyArray3D[x, y, 1] = value_z;
                break;
            case 3:
                if (flagCript)
                {
                    value_z = MyArray3D[x, y, 3];

                    MyArray3D[x, y, 3] = MyArray3D[x, y, 0];
                    MyArray3D[x, y, 0] = MyArray3D[x, y, 1];
                    MyArray3D[x, y, 1] = MyArray3D[x, y, 2];
                    MyArray3D[x, y, 2] = value_z;
                }
                else
                {
                    value_z = MyArray3D[x, y, 1];

                    MyArray3D[x, y, 1] = MyArray3D[x, y, 0];
                    MyArray3D[x, y, 0] = MyArray3D[x, y, 3];
                    MyArray3D[x, y, 3] = MyArray3D[x, y, 2];
                    MyArray3D[x, y, 2] = value_z;
                }
                break;
            default:
                break;
        }
    }

    public static string[] MyArrKeyF(string[] MyArrKey)
    {
        string strKey;

        int round = MyArrKey.Length;
        BitArray myList;
        bool[, ,] MyArray3D_new = new bool[4, 4, 4];
        int j, i, k = 0, x, y, z;
        byte[] randomKeyNew = new byte[kMax * round]; //полный ключ в байтах
        byte[] MyArr1D = new byte[kMax]; //выходной массив байт для одной строки
        BitArray MyArrBool = new BitArray(64);

        string[] MyStrKey = new string[round];
        string[] Text_KeyF = new string[round];

        for (int i_round = 0; i_round < round; i_round++)
        {
            strKey = MyArrKey[i_round];
            //преобразовываем строку из массива в массив байт
            for (j = 0, i = 0; i < strKey.Length; i += 2)
            {
                randomKeyNew[j++] = Convert.ToByte(strKey.Substring(i, 2), 16);
            }
            //преобразовываем массив байт в массив бит
            myList = new BitArray(randomKeyNew);

            //преобразовываем массив бит в трехмерный логический массив
            //Создадим НОВЫЙ 3-D массив bit
            for (x = 0; x < 4; x++)
```

```
                {
                    for (y = 0; y < 4; y++)
                    {
                        for (z = 0; z < 4; z++)
                        {
                            MyArray3D_new[y, z, x] = myList[k++];
                        }
                    }
                }

                //Перейдем от массива 3-D к одномерному массиву
                k = 0;
                for (x = 0; x < 4; x++)
                {
                    for (y = 0; y < 4; y++)
                    {
                        for (z = 0; z < 4; z++)
                        {
                            MyArrBool[k++] = MyArray3D_new[x, y, z];
                        }
                    }
                }

                //преобразование битов в байты
                MyArrBool.CopyTo(MyArr1D, 0);

                //только первые четыре байта масива копируем. Из них выйдет 8 символов
                MyStrKey = MyArr1D.Select(num => Convert.ToString(num, 16).PadLeft(2,
'0')).ToArray(); //Если строка из двух символов не получается, То добавляем 0

                for (k = 0; k < kMax / 2; k++) //берем половину ключа!!!!!!!!!!!!!!!!!!
                {
                    Text_KeyF[i_round] += MyStrKey[k].ToUpper(); //для ключа, беререм
первые 8-мь символов (4-ре двойных символа)
                }
                k = 0;
            }

            return Text_KeyF;
        }
    }
}
```

Програмний код класу “SlaideCode.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RSB_64_3D
{
    static class SlaideCode
    {
        static public bool flagCript;

        public static void SlaideCode_crypt(ref List<bool[, ,]> listFile, int i_round,
bool[, ,] arrKeyRound, int blokCub, bool flag_A) //возможно нужно будет добавиь return
List
        {
```

```
            bool[, ,] arrKeyCub = new bool[4, 4, 4]; //ключ для каждого кубика
            bool[, ,] arrFileCubKey = new bool[4, 4, 4]; //данные каждого кубика
            int listFile_Count = listFile.Count;
            arrKeyCub = arrKeyRound; //первоначальное значение ключа ???
            if (flagCript)
            {
                switch (i_round % 2)
                {
                    case 0: //нечетный раунд
                        for (int i_cub = 0; i_cub < listFile_Count; i_cub++)
                        {
                            cript_XOR(ref listFile, arrKeyCub, i_cub);

                            if (flag_A) arrKeyCub = listFile[i_cub]; //ключ для
следующего кубика при зашифровке
                            else //flag_B зашифровка нечетных раундов
                            {
                                if ((i_cub + 1) % blokCub == 0) arrKeyCub =
arrKeyRound; //первоначальное значение ключа ???
                                else arrKeyCub = listFile[i_cub]; //ключ для следующего
кубика при зашифровке
                            }
                        }
                        break;
                    default: //четный раунд
                        for (int i_cub = listFile_Count - 1; i_cub >= 0; i_cub--)
                        {
                            cript_XOR(ref listFile, arrKeyCub, i_cub);

                            if (flag_A) arrKeyCub = listFile[i_cub]; //ключ для
следующего кубика при зашифровке
                            else //flag_B зашифровка нечетных раундов
                            {
                                if ((i_cub - 1) % blokCub == 0) arrKeyCub =
arrKeyRound; //первоначальное значение ключа ???
                                else arrKeyCub = listFile[i_cub]; //ключ для следующего
кубика при зашифровке
                            }
                        }
                        break;
                }
            }
            else //расшифровка
            {
                switch (i_round % 2)
                {
                    case 0:
                        for (int i_cub = 0; i_cub < listFile_Count; i_cub++)
                        {
                            arrFileCubKey = copyArr(listFile[i_cub]);
                            cript_XOR(ref listFile, arrKeyCub, i_cub);

                            if (flag_A) arrKeyCub = arrFileCubKey; //ключ для
следующего кубика при расшифровке
                            else //flag_B
                            {
                                if ((i_cub + 1) % blokCub == 0) arrKeyCub =
arrKeyRound; //первоначальное значение ключа ???
```

```
                                else arrKeyCub = arrFileCubKey; //ключ для следующего
кубика при расшифровке
                            }
                        }
                        break;
                    default:
                        for (int i_cub = listFile_Count - 1; i_cub >= 0; i_cub--)
                        {
                            arrFileCubKey = copyArr(listFile[i_cub]);
                            cript_XOR(ref listFile, arrKeyCub, i_cub);

                            if (flag_A) arrKeyCub = arrFileCubKey; //ключ для
следующего кубика при расшифровке
                            else //flag_B зашифровка четных раундов
                            {
                                if ((i_cub - 1) % blokCub == 0) arrKeyCub =
arrKeyRound; //первоначальное значение ключа ???
                                else arrKeyCub = arrFileCubKey; //ключ для следующего
кубика при зашифровке
                            }
                        }
                        break;
                }
            }
        }

        public static void cript_XOR(ref List<bool[, ,]> listFile, bool[, ,] arrKeyCub,
int i_cub)
        {
            for (int z = 0; z < 4; z++)
            {
                for (int y = 0; y < 4; y++)
                {
                    for (int x = 0; x < 4; x++)
                    {
                        listFile[i_cub][z, y, x] ^= arrKeyCub[z, y, x]; // XOR
                    }
                }
            }
        }

        public static bool[, ,] copyArr(bool[, ,] arrCub)
        {
            bool[, ,] arrKey = new bool[4, 4, 4];
            for (int z = 0; z < 4; z++)
            {
                for (int y = 0; y < 4; y++)
                {
                    for (int x = 0; x < 4; x++)
                    {
                        arrKey[z, y, x] = arrCub[z, y, x];
                    }
                }
            }
            return arrKey;
        }
    }
}
```

Програмний код класу “SubCode_3D.cs”

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using System.Collections; //для BitArray и Hashtable

namespace RSB_64_3D
{
    static class SubCode_3D
    {
        const int kMax = 8; //размер кубика в байтах
        //static public bool flagCript;

        public static void SubCode_3D_cript(ref List<bool[, ,]> listFile, byte[,]
MyArrSBox_2D_A, byte uArr)
        {
            byte[] value; //= new byte[kMax];
            int i, j;
            //для 1-D преобразования
            byte vL; //младший полубайт
            byte vH; //старший полубайт
            byte v; //байт

            //начинаем саму обработку кубиков
            int iMax = listFile.Count; //количество кубиков в файле
            for (i = 0; i < iMax; i++)
            {
                value = Array_3D_to_1D(listFile[i], uArr); //получаем 1D байтовый
массив (из кубика)

                for (j = 0; j < kMax; j++)
                {
                    v = value[j];

                    if (v >= 0xF)
                    {
                        vH = (byte)(v >> 4);
                        vL = (byte)(((byte)(v << 4)) >> 4);
                    }
                    else
                    {
                        vH = 0;
                        vL = v;
                    }

                    //Извлекаем нужный байт из таблицы S-Box и записываем новое
значение
                    value[j] = MyArrSBox_2D_A[vH, vL]; //1D байтовый массив
                }
                //преобразовываем 1D byt[] в 3D bool[,,] и записываем в массив файла
                listFile[i] = Array_1D_to_3D(value, uArr);
            }

        }

        public static byte[] Array_3D_to_1D(bool[, ,] value64, byte uArr)
        {
```

```
            //Делаем выборку из 3D массива, согласно схеме кубиков. И собираем 1D
массив
            byte[] MyArr1 = new byte[kMax];
            BitArray MyArr2 = new BitArray(64);
            //bool[] MyArr2 = new bool[64];
            int i = 0, z, y, x;
            switch (uArr)
            {
                case 1:
                    for (z = 0; z < 4; z++)
                    {
                        for (y = 0; y < 4; y++)
                        {
                            for (x = 0; x < 4; x++)
                            {
                                MyArr2[i++] = value64[z, y, x];
                            }
                        }
                    }
                    break;
                case 2:
                    for (x = 0; x < 4; x++)
                    {
                        for (z = 0; z < 4; z++)
                        {
                            for (y = 0; y < 4; y++)
                            {
                                MyArr2[i++] = value64[z, y, x];
                            }
                        }
                    }
                    break;
                case 3:
                    for (y = 0; y < 4; y++)
                    {
                        for (x = 0; x < 4; x++)
                        {
                            for (z = 0; z < 4; z++)
                            {
                                MyArr2[i++] = value64[z, y, x];
                            }
                        }
                    }
                    break;
                default:
                    break;
            }

            //преобразование битов в байты
            MyArr2.CopyTo(MyArr1, 0);

            return MyArr1; //одномерный массив байт
        }

        public static bool[, ,] Array_1D_to_3D(byte[] value, byte uArr)
        {
            int i = 0, z, y, x;
            bool[, ,] MyArray3D = new bool[4, 4, 4];
```

```
            //Преобразуем массив byte в bit
            BitArray myList = new BitArray(value);
            switch (uArr)
            {
                case 1:
                    for (z = 0; z < 4; z++)
                    {
                        for (y = 0; y < 4; y++)
                        {
                            for (x = 0; x < 4; x++)
                            {
                                MyArray3D[z, y, x] = myList[i++];
                            }
                        }
                    }
                    break;
                case 2:
                    for (x = 0; x < 4; x++)
                    {
                        for (z = 0; z < 4; z++)
                        {
                            for (y = 0; y < 4; y++)
                            {
                                MyArray3D[z, y, x] = myList[i++];
                            }
                        }
                    }
                    break;
                case 3:
                    for (y = 0; y < 4; y++)
                    {
                        for (x = 0; x < 4; x++)
                        {
                            for (z = 0; z < 4; z++)
                            {
                                MyArray3D[z, y, x] = myList[i++];
                            }
                        }
                    }
                    break;
                default:
                    break;
            }
            return MyArray3D; //трехмерный массив bool
        }
    }
}
```

ДОДАТОК 4. Програмний код шифру M3DCrypt

Програмний код головного вікна

```
using System;
using System.ComponentModel;
using System.IO;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
using System.Threading.Tasks;
using System.Windows;
using StreamCipher.Controls;
using StreamCipher.Controls.Model;

namespace StreamCipher
{
    public unsafe class MainWindowViewModel : INotifyPropertyChanged
    {
        private string _workTime;
        public string WorkTime
        {
            get { return _workTime; }
            set
            {
                _workTime = value;
                OnPropertyChanged();
            }
        }

        private int _progress;
        public int Progress
        {
            get { return _progress; }
            set
            {
                _progress = value;
                WorkTime = (DateTime.UtcNow - _startDateTime).ToString();
                OnPropertyChanged();
            }
        }

        private Visibility _progressVisibility;
        public Visibility ProgressVisibility
        {
            get { return _progressVisibility; }
            set
            {
                _progressVisibility = value;
                OnPropertyChanged();
            }
        }


        public CipherSettingsViewModel CipherSettingsViewModel { get; }
        public FilesViewModel FilesViewModel { get; }

        public MainWindowViewModel()
        {
            FilesViewModel = new FilesViewModel();
```

```
            CipherSettingsViewModel = new CipherSettingsViewModel();
            ProgressVisibility = Visibility.Collapsed;
        }

        public void CodedAsync()
        {
            Task.Factory.StartNew(Coded);
        }

        private uint* _currentSatate;
        private byte* _sbox0, _sbox1, _sbox2, _sbox3;//, _sbox4, _sbox5, _sbox6,
_sbox7;
        private byte* _index0, _index1, _index2, _index3;
        private DateTime _startDateTime;
        public void Coded()
        {
            if(!FilesViewModel.FilesIsValid())
                return;

            _startDateTime = DateTime.UtcNow;
            Progress = 0;
            ProgressVisibility = Visibility.Visible;

            try
            {
                allocMemoryAndInitCoder();

                File.Create(FilesViewModel.OutputFileName).Close();
                using (var fileWrite = new
BinaryWriter(File.Open(FilesViewModel.OutputFileName, FileMode.Open,
FileAccess.Write)))
                using (var fileRead = new
BinaryReader(File.Open(FilesViewModel.InputFileName, FileMode.Open, FileAccess.Read)))
                {
                    const int readBufer = 3200000;
                    while (fileRead.BaseStream.Position != fileRead.BaseStream.Length)
                    {
                        var temp = fileRead.ReadBytes(readBufer);
                        Progress = (int) (((double)
fileRead.BaseStream.Position/fileRead.BaseStream.Length)*100);

                        codedBytes(ref temp);
                        fileWrite.Write(temp);
                    }
                }
            }
            catch (IOException)
            {
            }
            finally
            {
                freeMemory();
            }

            ProgressVisibility = Visibility.Collapsed;
            FilesViewModel.RecalcOutputFileEntropy();
        }

        private void allocMemoryAndInitCoder()
```

```
        {
            var len = CipherSettingsViewModel.Sboxes[0].ArrayBytes.Length;
            _sbox0 = (byte*)Marshal.AllocHGlobal(len);
            _sbox1 = (byte*)Marshal.AllocHGlobal(len);
            _sbox2 = (byte*)Marshal.AllocHGlobal(len);
            _sbox3 = (byte*)Marshal.AllocHGlobal(len);


            Marshal.Copy(CipherSettingsViewModel.Sboxes[0].ArrayBytes, 0,
(IntPtr)_sbox0, len);
            Marshal.Copy(CipherSettingsViewModel.Sboxes[1].ArrayBytes, 0,
(IntPtr)_sbox1, len);
            Marshal.Copy(CipherSettingsViewModel.Sboxes[2].ArrayBytes, 0,
(IntPtr)_sbox2, len);
            Marshal.Copy(CipherSettingsViewModel.Sboxes[3].ArrayBytes, 0,
(IntPtr)_sbox3, len);

            _currentSatate = (uint*)Marshal.AllocHGlobal(sizeof(uint));
            *_currentSatate = (uint)(CipherSettingsViewModel.InitBytesRegister[3] << 24
|
                                     CipherSettingsViewModel.InitBytesRegister[2] << 16
|
                                     CipherSettingsViewModel.InitBytesRegister[1] << 8
|
                                     CipherSettingsViewModel.InitBytesRegister[0]);

            _index0 = (byte*)_currentSatate;
            _index1 = _index0 + 1;
            _index2 = _index0 + 2;
            _index3 = _index0 + 3;
        }
        private void freeMemory()
        {
            Marshal.FreeHGlobal((IntPtr)_sbox0);
            Marshal.FreeHGlobal((IntPtr)_sbox1);
            Marshal.FreeHGlobal((IntPtr)_sbox2);
            Marshal.FreeHGlobal((IntPtr)_sbox3);
            Marshal.FreeHGlobal((IntPtr)_currentSatate);
        }

        private void codedBytes(ref byte[] arr)
        {
            fixed (byte* bytes = arr)
                for (byte* bytePtr = bytes, end = bytes + arr.Length; bytePtr < end;
bytePtr++, (*_currentSatate)++)
                {
                    var b0 = *(_sbox0 + *_index0) ^ *(_sbox1 + *_index1) ^ *(_sbox2 +
*_index2) ^ *(_sbox3 + *_index3);
                    var b1 = *(_sbox0 + *_index3) ^ *(_sbox1 + *_index0) ^ *(_sbox2 +
*_index1) ^ *(_sbox3 + *_index2);
                    var b2 = *(_sbox0 + *_index2) ^ *(_sbox1 + *_index3) ^ *(_sbox2 +
*_index0) ^ *(_sbox3 + *_index1);
                    var b3 = *(_sbox0 + *_index1) ^ *(_sbox1 + *_index2) ^ *(_sbox2 +
*_index3) ^ *(_sbox3 + *_index0);

                    *bytePtr ^= (byte)(*(_sbox0 + b0) ^ *(_sbox1 + b1) ^ *(_sbox2 + b2)
^ *(_sbox3 + b3));
                }
        }
```

```
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName
= null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Програмний код налаштувань шифратору

```
using System.Windows;
using System.Windows.Controls;
using StreamCipher.Controls.Model;

namespace StreamCipher.Controls
{
    public partial class CipherSettings
    {
        public static DependencyProperty ViewModelProperty =
DependencyProperty.Register("ViewModel", typeof(CipherSettingsViewModel),
typeof(CipherSettings));
        public CipherSettingsViewModel ViewModel
        {
            get { return (CipherSettingsViewModel)GetValue(ViewModelProperty); }
            set { SetValue(ViewModelProperty, value);}
        }

        public CipherSettings()
        {
            InitializeComponent();
        }

        private void genereteBytesForRegisterOnClick(object sender, RoutedEventArgs e)
        {
            ViewModel.GenereteNewBytesForRegister();
        }

        private void registerOnTextChanged(object sender, TextChangedEventArgs e)
        {
            var textBox = sender as TextBox;
            if(textBox == null)
                return;

            ViewModel.SetInitBytesRegister(textBox.Text);
        }

        private void generateNewSboxOnClick(object sender, RoutedEventArgs e)
        {
            ViewModel.GenerateNewSbox();
        }
    }
}
```

Програмний код інтерфейсу шифратору

```
using System;
using System.ComponentModel;
using System.Globalization;
```

```
using System.Runtime.CompilerServices;
using System.Threading.Tasks;
using System.Windows;
using Microsoft.Win32;
using StreamCipher.Annotations;
using StreamCipher.Infrastructure;

namespace StreamCipher.Controls
{
    public partial class FilesView
    {
        public static DependencyProperty FilesViewModelProperty =
DependencyProperty.Register("ViewModel", typeof(FilesViewModel), typeof(FilesView));
        public FilesViewModel ViewModel
        {
            get { return (FilesViewModel)GetValue(FilesViewModelProperty); }
            set { SetValue(FilesViewModelProperty, value);}
        }

        public FilesView()
        {
            InitializeComponent();
        }

        private void setInputFileOnClick(object sender, RoutedEventArgs e)
        {
            var openFileDialog = new OpenFileDialog();
            if (openFileDialog.ShowDialog() != true)
                return;

            ViewModel?.SetInputFileName(openFileDialog.FileName);
        }

        private void setOutputFileOnClick(object sender, RoutedEventArgs e)
        {
            var saveFileDialog = new SaveFileDialog();
            if (saveFileDialog.ShowDialog() != true)
                return;

            ViewModel?.SetOutputFileName(saveFileDialog.FileName);
        }
    }

    public class FilesViewModel : INotifyPropertyChanged
    {
        private string _inputFileEntropy;
        public string InputFileEntropy
        {
            get { return _inputFileEntropy; }
            private set
            {
                _inputFileEntropy = value;
                OnPropertyChanged();
            }
        }

        private string _outputFileEntropy;
        public string OutputFileEntropy
        {
```

```
            get { return _outputFileEntropy; }
            private set
            {
                _outputFileEntropy = value;
                OnPropertyChanged();
            }
        }

        private string _inputFileName;
        public string InputFileName
        {
            get { return _inputFileName; }
            private set
            {
                _inputFileName = value;
                OnPropertyChanged();
            }
        }

        private string _outputFileName;
        public string OutputFileName
        {
            get { return _outputFileName; }
            private set
            {
                _outputFileName = value;
                OnPropertyChanged();
            }
        }

        private string _inputFileSize;
        public string InputFileSize
        {
            get { return _inputFileSize; }
            private set
            {
                _inputFileSize = value;
                OnPropertyChanged();
            }
        }

        public void SetOutputFileName(string fileName)
        {
            OutputFileName = fileName;
        }

        public void SetInputFileName(string fileName)
        {
            InputFileName = fileName;
            InputFileSize = FileSizeInfo.Info(fileName).ShortForm;
            RecalcInputFileEntropy();
        }

        public async void RecalcInputFileEntropy()
        {
            InputFileEntropy = await calc(InputFileName, progress => InputFileEntropy =
progress);
        }
        public async void RecalcOutputFileEntropy()
```

```
        {
            OutputFileEntropy = await calc(OutputFileName, progress =>
OutputFileEntropy = progress);
        }
        private static Task<string> calc(string fileName, Action<string> progress)
        {
            return Task<string>.Factory.StartNew(() =>
            {
                try
                {
                    return Entropy.Value(fileName, i =>
progress($"{i}%...")).ToString(CultureInfo.InvariantCulture);
                }
                catch (Exception)
                {
                    // ignored
                }
                return string.Empty;
            });
        }

        public bool FilesIsValid()
        {
            if (string.IsNullOrEmpty(InputFileName))
            {
                MessageBox.Show("Необходимо указать входной файл.", "",
MessageBoxButton.OK, MessageBoxImage.Information);
                return false;
            }
            if (string.IsNullOrEmpty(OutputFileName))
            {
                MessageBox.Show("Необходимо указать выходной файл.", "",
MessageBoxButton.OK, MessageBoxImage.Information);
                return false;
            }
            return true;
        }

        public event PropertyChangedEventHandler PropertyChanged;
        [NotifyPropertyChangedInvocator]
        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName
= null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

Клас «Entropy.cs»

```
using System;
using System.IO;
using System.Threading;

namespace StreamCipher.Infrastructure
{
    internal static class Entropy
    {
        public static double Value(string file, Action<int> progress)
        {
```

```
            ulong[] arr;
            return Calculate(file, progress, out arr);
        }

        public static ulong[] Calculate(string file, Action<int> progress)
        {
            ulong[] masCountBytes = new ulong[256];
            try
            {
                using (var fileRead = new BinaryReader(File.Open(file, FileMode.Open,
FileAccess.Read)))
                {
                    var length = fileRead.BaseStream.Length;

                    while (fileRead.BaseStream.Position != length)
                    {
                        byte[] temp1 = fileRead.ReadBytes(3200000);

                        progress((int)(((double)fileRead.BaseStream.Position /
fileRead.BaseStream.Length) * 100));

                        foreach (byte b in temp1)
                            masCountBytes[b]++;
                    }
                }
            }
            catch (IOException) { }

            return masCountBytes;
        }

        public static double Calculate(string file, Action<int> progress, out ulong[]
masBytes)
        {
            masBytes = Calculate(file, progress);

            long length = 1;
            try
            {
                length = new FileInfo(file).Length;
            }
            catch (IOException) { }

            double entropy = 0;
            for (int i = 0; i < masBytes.Length; i++)
                if (masBytes[i] != 0)
                {
                    double temp = (double) masBytes[i]/length;
                    entropy += -temp * Math.Log(temp, 2);
                }
            progress(100);
            Thread.Sleep(300);
            return entropy;
        }
    }
}
```

Клас «SizeInfo.cs»

```
using System.IO;
```

```
namespace StreamCipher.Infrastructure
{
    internal class FileSizeInfo
    {
        public FileSizeInfo(string fileName)
        {
            var fileInfo = new FileInfo(fileName);
            Size = fileInfo.Length;
        }
        public string ShortForm
        {
            get
            {
                string[] arr = {" ", " K", " M", " G"};
                double size = Size;

                int index = 0;
                while (size > 1024 && index < 4)
                {
                    size /= 1024;
                    index++;
                }
                return size.ToString("f2") + arr[index] + "b (" + Size.ToString("###
### ### ### ###").TrimStart(' ') + " байт)";
            }
        }

        public long Size { get; }

        public static FileSizeInfo Info(string fileName)
        {
            return new FileSizeInfo(fileName);
        }
    }
}
```

Клас «GenerateBytesSequense.cs»

```
using System.Security.Cryptography;

namespace StreamCipher.Infrastructure
{
    static class GenerateBytesSequense
    {
        public static byte[] Get(int byteCount)
        {
            byte[] arr = new byte[byteCount];
            new RNGCryptoServiceProvider().GetNonZeroBytes(arr);
            return arr;
        }
    }
}
```

ДОДАТОК 5. Акти впровадження у навчальний процес

З А Т В Е Р Д Ж У Ю

Т.в.о. проректора з наукової роботи
Національного авіаційного університету

_____ О.К. Юлін
«13» 02 2017 р.

АКТ ВПРОВАДЖЕННЯ

результатів дисертаційної роботи здобувача наукового ступеня кандидата технічних наук Навроцького Д.О. у навчальний процес Національного авіаційного університету

Ми, що нижче підписалися, *завідувач кафедри електроніки Навчально-наукового інституту аеронавігації д.т.н., проф. Ф.Й. Яновський, директор Навчально-наукового інституту аеронавігації Національного авіаційного університету д.т.н., проф. І.О. Мачалін*
склали цей акт про те, що результати наукових досліджень *за темою дисертаційної роботи на здобуття наукового ступеня кандидата технічних наук Навроцького Дениса Олександровича «Метод побудови симетричних криптографічних шифрів на основі тривимірних керованих перетворень»*
використовуються у навчальному процесі *на кафедрі електроніки Навчально-наукового інституту аеронавігації Національного авіаційного університету.*

Найменування впровадженого результату	Форма впровадження і досягнутий фактичний ефект
Програмні і апаратне забезпечення для формування керованих криптографічних примітивів	*У навчальний процес як програмне і апаратне забезпечення для проведення лабораторних занять з дисциплін «Основи криптографії» та «Новітні технології захисту інформації». Завдяки своїй наочності, можливості оперативного проведення модифікацій програмних модулів та інтеграції з апаратними засобами ЕОМ, впроваджені програмно-апаратні засоби підвищили ефективність засвоєння студентами матеріалу зазначених дисциплін.*
Програмно-апаратна реалізація криптографічного захисту каналу зв'язку БПЛА для командної і телеметричної інформації.	*У навчальний процес як програмне і апаратне забезпечення для проведення лабораторних та практичних занять з дисциплін «Алгебраїчні основи теорії кодування» та «Теорія інформації та кодування». Завдяки своїй наочності, дозволив підвищили якість підготовки студентів, що навчаються за спеціальністю 6.050801 «Мікро- та наноелектроніка" і 6.050802 "Електронні пристрої та системи" шляхом набуття ними знань та вмінь проектувати і реалізовувати захищений канал зв'язку для командної і телеметричної інформації БПЛА типу «земля-борт-земля».*

Завідувач кафедри
електроніки,
д.т.н., проф. _____ Ф.Й. Яновський

Директор Навчально-наукового
Інституту аеронавігації,
д.т.н., проф. _____ І.О. Мачалін

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського”

03056, Київ-56, Проспект Перемоги, 37. тел. (+38 044) 236-79-89

Вих № 6 від 15.02.2017р.

Приладобудівний факультет

З А Т В Е Р Д Ж У Ю

Декан приладобудівного факультету

_____ Г.С. Тимчик

«_____» __лютого__ 2017 р.

АКТ ВПРОВАДЖЕННЯ

результатів дисертаційної роботи здобувача наукового ступеня кандидата технічних наук Навроцького Д.О. у навчальний процес НТУУ «КПІ ім. Ігоря Сікорського»

Ми, що нижче підписалися, *в. о. завідувача кафедри виробництва приладів приладобудівного факультету к.т.н., доцент В.В.Шевченко, доценти кафедри виробництва приладів к.т.н., доц. М.Ф. Терещенко, к.т.н. доц. С.П. Вислоух*

склали цей акт про те, що результати наукових досліджень *за темою дисертаційної роботи на здобуття наукового ступеня кандидата технічних наук Навроцького Дениса Олександровича «Метод побудови симетричних криптографічних шифрів на основі тривимірних керованих перетворень»*

використовуються у навчальному процесі *на кафедрі виробництва приладів приладобудівного факультету Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорськоно»*

Найменування впровадженого результату	**Форма впровадження і досягнутий фактичний ефект**
Програмні і апаратне забезпечення для формування керованих криптографічних примітивів	*У навчальний процес, як програмне і апаратне забезпечення, для захисту медичних даних від несанкціонованого доступу при проведенні лабораторних занять з дисциплін «Генезис біосигналів» та «Акустичні медичні прилади». Завдяки своїй наочності, можливості оперативного проведення модифікацій програмних модулів та інтеграції з апаратними засобами ЕОМ, впроваджені програмно-апаратні засоби підвищили ефективність захисту медичних даних від несанкціонованого доступу і засвоєння студентами матеріалу зазначених дисциплін, а це дало можливість вивчати різні способи захисту медичної інформації при передачі і зберіганні*
Програмно-апаратна реалізація криптографічного захисту каналу зв'язку при передачі медичної телеметричної інформації.	*У навчальний процес, як програмне і апаратне забезпечення для проведення занять з дисципліни «Біометрія». Завдяки своїй наочності, дозволив підвищили якість підготовки студентів, що навчаються за сп. іальністю 8.05100307 «Медичні прилади та системи» шляхом набуття ними знань та вмінь по захисту каналу зв'язку при передачі даних від периферії (датчиків) медичного приладу до ПК, та зберігання конфіденційних (медичних) даних пацієнта в зашифрованому вигляді.*

В.о. завідувача кафедри
виробництва приладів, к.т.н., доцент _____ В.В. Шевченко

Доцент кафедри
виробництва приладів , к.т.н., доцент _____ М.Ф. Терещенко

Доцент кафедри
виробництва приладів ,к.т.н., доцент _____ С.П. Вислоух

ДОДАТОК 6. Акти впровадження в виробництво

Personal Communications GRATIS,Ltd

ИЗМЕРИТЕЛЬНАЯ ЛАБОРАТОРИЯ
Аттестат аккредитации УкрЦСМ на техническую компетентность № ПТ-131/13 от 24.04.13
AUTHORIZERED SERVICE CENTER MOTOROLA, ALCATEL, SAMSUNG, TEKTRONIX, ARPEGE, HAMEG, PMR, SIT, RSSYS, GEDIS, ROHDE&SCHWARZ
г. Киев, ул. Бажова, 15/20, ☎ (044)573-2603 (многоканальный) (информация о ремонтах), www.gratis.com.ua, email: service@gratis.com.ua

«УТВЕРЖДАЮ»
Директор ООО «ГРАТИС, Лтд»
А.П. Кириченко
«13» 02 2017 г.

А К Т

Внедрения результатов диссертационной работы
НАВРОЦКОГО Дениса Александровича

«МЕТОД ПОСТРОЕНИЯ СИММЕТРИЧНЫХ КРИПТОГРАФИЧЕСКИХ ШИФРОВ НА ОСНОВЕ ТРЕХМЕРНЫХ УПРАВЛЯЕМЫХ ПРЕОБРАЗОВАНИЙ»

Комиссия в составе президента ООО «ГРАТИС, Лтд» к.т.н., доцента Кириченко Е.П., и главного инженера Газизова Р.Э. составила накстоящий акт о том, что в приод 2015-2017 годах в научных исследованиях и разработках Специализированного центра «Персональные коммуникации» и измерительной лаборатории компании использовались результаты диссертационной работы Навроцкого Дениса Александровича «Метод построения симметричных криптографических шифров на основе трехмерных управляемых преобразований» для закрытия каналов обмена информацией в логистических сетях доступа к базам производителей РЭА на основе разработанного автором диссертации криптографического алгоритма.

Президент ООО «ГРАТИС, Лтд»,
к.т.н., доцент Е. Кириченко

Главный инженер
ООО «ГРАТИС, Лтд» Р. Газизов

ТОВ «АГФАР»

03164, м. Київ-164, вул. Булаховського, 5-б, оф.10, ЄДРПОУ 24087632

тел./факс (044) 409-14-88, 409-29-44, 423-67-26 E-mail: agfar@ukr.net

«ЗАТВЕРДЖУЮ»

Директор ТОВ «АГФАР»

_____ В.Ф. Терещенко

«_07_»__лютого__ 2017 р.

А К Т

впровадження результатів дисертаційної роботи

НАВРОЦЬКОГО Дениса Олександровича

«МЕТОД ПОБУДОВИ СИМЕТРИЧНИХ КРИПТОГРАФІЧНИХ ШИФРІВ НА ОСНОВІ ТРИВИМІРНИХ КЕРОВАНИХ ПЕРЕТВОРЕНЬ»

Даний акт складено про те, що результати дисертаційної роботи Навроцького Дениса Олександровича «Метод побудови симетричних криптографічних шифрів на основі тривимірних керованих перетворень» впроваджено та використано у діяльності ТОВ "АГФАР".

Під час написання дисертації Навроцький Д.О. розробив інженерне програмне забезпечення для криптографічного захисту інформації.

Програмний засіб, призначений для захисту інформації, може бути використаним під час зберігання і транспортуванні даних, що є комерційною таємницею, а також для роботи з важливими документами, доступ до яких може отримати третя особа.

У діяльності нашої компанії даний засіб використано для захисту технічної і технологічної документації.

Результати отримані Навроцьким Д.О. під час написання дисертаційної роботи, дозволили зменшити витрати часу, трудових та грошових ресурсів для захисту цінних даних.

Заст. директор
к.т.н., доцент М.Ф. Терещенко

Технічний директор В.А. Єфім'єв

Головний інженер О. В. Маляр

САЙФЕР
Системи захисту інформації

Адреса: 04107, Київ, вул. Нагірна, 25
Тел./Факс: (044) 484-46-17, 484-46-12, 483-03-22
E-mail: sales@cipher.kiev.ua
http://www.elpay.com; http://www.cipher.kiev.ua

АКТ

про впровадження результатів дисертаційної роботи
Навроцького Дениса Олександровича
«МЕТОД ПОБУДОВИ СИМЕТРИЧНИХ КРИПТОГРАФІЧНИХ ШИФРІВ НА ОСНОВІ ТРИВИМІРНИХ КЕРОВАНИХ ПЕРЕТВОРЕНЬ»

Комісія у складі голови – директора товариства з обмеженою відповідальністю «Сайфер ЛТД», кандидата технічних наук Боровікова О.М., членів комісії – провідного розробника Бойко С.Т., провідного розробника Прокоповича Л.Ю., складено цей акт про те, що при розробці бібліотек криптографічних примітивів «Шифр+» v2.1, реалізовано такі результати наукових досліджень Навроцького Дениса Олександровича:

Використано симетричний шифр на основі тривимірних керованих перетворень, що дозволило підвищити швидкодію до 13%, в залежності від довжини блоку перетворень.

Запропоновані, у результаті виконання наукових досліджень Навроцького Дениса Олександровича, алгоритми, дозволяють суттєво підвищити швидкодію криптографічних перетворень «Шифр+» v.2.1 у компонентах центру сертифікації ключів на основі системи криптографічного захисту інформації «Шифр-X.509».

Голова комісії
Директор ТОВ «Сайфер ЛТД», к.т.н. О.М. Боровіков

Члени комісії:

Провідний розробник С.Т. Бойко

Провідний розробник Л.Ю. Прокопович