ІНСТИТУТ СПЕЦІАЛЬНОГО ЗВ`ЯЗКУ ТА ЗАХИСТУ ІНФОРМАЦІЇ
НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

На правах рукопису

СТОРОЖУК АРТЕМ ЮРІЙОВИЧ

УДК 621.391:519.2:510.5

МЕТОДИ ОЦІНЮВАННЯ ТА ОБГРУНТУВАННЯ СТІЙКОСТІ ПОТОКОВИХ ШИФРІВ ВІДНОСНО СТАТИСТИЧНИХ АТАК НА ОСНОВІ АЛГЕБРАЇЧНО ВИРОДЖЕНИХ НАБЛИЖЕНЬ БУЛЕВИХ ФУНКЦІЙ

21.05.01 – інформаційна безпека держави

Дисертація на здобуття наукового ступеня
кандидата технічних наук

Науковий керівник:
кандидат технічних наук, доцент
КОНЮШОК Сергій Миколайович

Київ – 2016

ЗМІСТ

ВСТУП

Актуальність теми. Найважливішою задачею забезпечення інформаційної безпеки держави є створення та підтримання умов, які гарантують безпечну обробку, зберігання та передачу державних інформаційних ресурсів [1, 2]. Для вирішення цієї задачі, поряд з технічними та організаційними, застосовують криптографічні методи, які базуються на спеціальних математичних перетвореннях інформації, що захищається. На даний час без криптографічних методів та засобів неможливо вирішення таких задач інформаційної безпеки як забезпечення конфіденційності, цілісності, автентичності, невідстежуваності (анонімності), неможливості відмови від авторства та ін. [3, 4]. Тому рівень розвитку криптографічних методів, які використовуються для захисту державних інформаційних ресурсів, суттєво впливає на інформаційну безпеку держави в цілому, її економічну незалежність та суверенітет.

Важливою складовою забезпечення безпеки інформації в сучасних спеціальних інформаційно-телекомунікаційних системах є синхронні потокові шифри. Це обумовлено надвисокою швидкістю потокового шифрування (зокрема, програмні реалізації слово-орієнтованих потокових шифрів є в 5 – 10 разів швидше в порівнянні з відповідними реалізаціями блокових шифрів [5]). На сьогодні лише потоковий шифр може забезпечити прийнятну швидкість шифрування на канальному, мережевому та транспортному рівнях. Програмно-орієнтовані потокові шифри знаходять застосування у вбудованих додатках систем з обмеженою кількістю обчислювальних ресурсів, зокрема, у бездротовій телефонії (шифр SOBER [6]) та в системах платного телебачення (шифр Panama [7]). Окремо варто відзначити алгоритм потокового шифрування A5/1, який є невід’ємною частиною стандарту мобільного зв’язку GSM [8].

Значний інтерес до потокових шифрів спостерігається у європейському науковому криптографічному співтоваристві. Про це свідчить проведення

міжнародних конкурсів NESSIE (2000 – 2003 рр.) та eSTREAM (2004 –2008 рр.), спрямованих на створення стійких потокових шифрів, придатних до широкого застосування. Через відсутність на даний час національного стандарту потокового шифрування в Україні вирішення цієї задачі є дуже актуальним і для нашої держави.

Найважливішою вимогою до потокових шифрів є умова їх практичної стійкості відносно всіх відомих криптоаналітичних атак. Якщо ця вимога не виконується, то інші характеристики шифру втрачають своє значення. В той же час, отримання науково обґрунтованих оцінок стійкості потокових шифрів (навіть відносно конкретних, добре вивчених методів криптоаналізу) є складною науковою проблемою. Фактично висновок про стійкість будь-якого потокового шифру ґрунтується на неможливості провести на нього окремі атаки, відомі криптоаналітикам, а також на припущенні про те, що майбутні атаки не призведуть до помітних покращень відомих криптоаналітичних методів. У зв’язку з цим особливу гостроту набуває проблема обґрунтування стійкості потокових шифрів, призначених для захисту інформації в спеціальних інформаційно-телекомунікаційних системах України.

Аналіз доступних наукових публікацій показує, що найбільш потужними та широко розповсюдженими на практиці є атаки на синхронні потокові шифри на основі підібраних векторів ініціалізації. До них відносяться зокрема, кубічна атака [9], атака Фішера-Хазаі-Маєра (FKM) [10], а також їх різноманітні модифікації та вдосконалення [11 – 16]. Як показує детальний аналіз цих атак, всі вони будуються на основі наближень булевих функцій, пов’язаних з алгоритмами шифрування, функціями менш складної структури. Обґрунтування практичної стійкості потокового шифру відносно будь-якої з цих атак зводиться до встановлення факту відсутності зазначених наближень.

Не дивлячись на помітний прогрес у розробці конкретних атак, слід констатувати, що низка важливих окремих наукових задач залишається невирішеною. Перш за все, це – відносна вузькість класів функцій-наближень,

які використовуються (афінні; функції від малого числа змінних). По друге, це – відсутність (за виключенням окремих спеціальних випадків [10, 17 – 24]) ефективних алгоритмів пошуку наближень булевих функцій із зазначених класів. Нарешті, відсутні методи, які б дозволяли за достатньо загальних умов обґрунтовувати стійкість синхронних потокових шифрів відносно зазначених вище атак.

Наведені факти свідчать про певне протиріччя між потребами в обґрунтовано стійких та практичних алгоритмах потокового шифрування, що необхідні для забезпечення безпеки державних інформаційних ресурсів, з одного боку, та відсутністю методів обґрунтування стійкості цих алгоритмів відносно широкого кола сучасних атак, з іншого. Зазначене протиріччя породжує *наукову задачу, яка полягає в розробці методів побудови науково обґрунтованих оцінок стійкості потокових шифрів відносно статистичних атак, що базуються на алгебраїчно вироджених наближеннях булевих функцій,* розв’язанню якої присвячено дану дисертаційну роботу.

Зв'язок роботи з науковими програмами, планами, темами. Робота над дисертацією проводилася відповідно до планів науково-дослідної роботи Інституту спеціального зв’язку та захисту інформації Національного технічного університету України “Київський політехнічний інститут” та в рамках науково-дослідних робіт (НДР) “Севрюга” (№ держреєстрації 0113U005813) та “Мокрель” (№ держреєстрації 0115U004118) на замовлення Служби зовнішньої розвідки України.

Мета та задачі досліджень. *Метою дисертаційної роботи є підвищення ефективності використання національних інформаційних ресурсів за рахунок зменшення часу проведення експертних досліджень алгоритмів потокового шифрування, призначених для захисту інформації в спеціальних інформаційно-телекомунікаційних системах України.*

Для досягнення поставленої мети в дисертаційній роботі сформульовано та вирішено наступні взаємозв’язані *окремі задачі досліджень*.

1. Проведено аналіз сучасних методів оцінювання та обґрунтування стійкості синхронних потокових шифрів відносно відомих статистичних атак на основі підібраних векторів ініціалізації.

2. Запропоновано статистичну атаку на фільтрувальній генератор гами з лінійним законом реініціалізації початкового стану та функцією ускладнення, що є близькою до алгебраїчно виродженої.

3. Запропоновано статистичну атаку на синхронні потокові шифри, яка узагальнює аналогічні раніше відомі атаки та має більшу ефективність в порівнянні з ними.

4. Розроблено метод побудови списку усіх високоймовірних k-вимірних наближень булевих функцій, що використовуються у синхронних потокових шифрах.

5. Розроблено метод оцінювання відносної відстані між зрівноваженою булевою функцією, заданою за допомогою оракула, та множиною k-вимірних функцій.

6. Розроблено метод пошуку алгебраїчно вироджених наближень булевих функцій для оцінювання стійкості синхронних потокових шифрів відносно узагальненої статистичної атаки.

7. Створено пакет прикладних програм для оцінювання та обгрунтування стійкості синхронних потокових шифрів відносно запропонованих статистичних атак.

8. Оцінено практичну стійкість шифру SNOW 2.0 відносно узагальненої статистичної атаки.

Об'єктом дослідження в дисертаційній роботі є криптографічні перетворення інформації, що реалізуються синхронними потоковими шифрами, призначеними для захисту інформації в інформаційно-телекомунікаційних системах України.

Предметом дослідження – методи оцінювання та обґрунтування стійкості синхронних потокових шифрів відносно статистичних атак на основі алгебраїчно вироджених наближень булевих функцій.

Методи дослідження. Основу дисертаційних досліджень складають теоретичні дослідження (математичні методи оцінювання та обґрунтування стійкості синхронних потокових шифрів). Для розв'язання окремих задач 2 – 6 застосовувалися методи теорії булевих функцій, гармонічного аналізу, лінійної алгебри і теорії ймовірностей; для розв'язання окремих задач 4, 5, 8 використовувалися також методи математичної статистики. Чисельні розрахунки на ПК виконувалися з використанням платформи .NET Framework 4.0, а також пакету прикладних програм Maple 15. Обробка статистичних даних в процесі дослідження проводилася у відповідності до положень математичної статистики.

Наукова новизна отриманих результатів. Підсумком вирішення перелічених вище окремих задач є такі нові наукові результати, що висуваються на захист.

1. Вдосконалено низку статистичних атак на синхронні потокові шифри, шляхом їхнього узагальнення та уніфікації, зокрема, атаку FKM та кубічну атаку [9, 10, 25 – 28]. Розроблені атаки базуються на алгебраїчно вироджених наближеннях булевих функцій. Отримані аналітичні оцінки трудомісткості запропонованих атак свідчать про їх більш високу ефективність у порівнянні з аналогічними раніше відомими. Зокрема, застосування однієї з цих атак до редукованої версії шифру Grain-128 дозволяє зменшити обчислювальну складність відновлення ключа шифру майже в 2^{27} разів у порівнянні з атакою FKM.

2. Вдосконалено раніше відомий алгоритм П. Гопалана [24] призначений для вирішення задачі побудови високоймовірних k-вимірних наближень булевих функцій. Запропонований метод розв'язання вказаної задачі має суттєво меншу трудомісткість у порівнянні з зазначеним алгоритмом, у певних

випадках в 1000 та більше разів. Зменшення трудомісткості запропонованого методу в порівнянні з [24] досягається за рахунок застосування більш точної оцінки кількості шуканих наближень вхідної функції, а також більш економної організації обчислень, яка базується на детальному аналізі структури цих наближень.

3. Вперше запропоновано метод обчислення значень нижніх меж відносної відстані між зрівноваженою булевою функцією та множиною всіх k-вимірних функцій від n змінних. Сутність методу полягає у статистичному оцінюванні відносної відстані за допомогою спеціально розробленого ймовірнісного алгоритму, складність якого залежить лінійно від n та поліноміально від величин, обернених до точності та ймовірності помилки алгоритму. Показано, що при малих значеннях k запропонований метод може бути ефективно використаний на практиці для обґрунтування стійкості функцій ускладнення синхронних потокових шифрів відносно узагальненої статистичної атаки.

4. Отримав подальший розвиток метод пошуку алгебраїчно вироджених наближень булевих функцій. Запропонований метод відрізняється за сутністю від відомих [10, 23, 24, 29] та базується на отриманих аналітичних умовах, яким задовольняють шукані наближення. На відміну від [23, 24], запропонований метод не має передумовою виконання будь-яких обмежень стосовно відстані, на якій треба відшукати наближення, а на відміну від [10, 29] він дозволяє знаходити наближення з більш широкого класу булевих функцій. Крім того, за певних умов запропонований метод надає можливість переконуватися у відсутності зазначених наближень, що дозволяє використовувати його для обґрунтування практичної стійкості синхронних потокових шифрів відносно відомих статистичних атак [10, 25 – 28].

Практичне значення отриманих результатів. Представлені в дисертаційній роботі нові наукові та практичні результати дозволяють:

– розширити клас наближень булевих функцій, що можуть бути використані для побудови статистичних атак на синхронні потокові шифри;

– встановити науково обґрунтовані умови стійкості синхронних потокових шифрів відносно відомих та запропонованих статистичних атак;

– підвищити (у певних випадках – в 1000 та більше разів) трудомісткість найкращого з відомих алгоритмів побудови високоймовірних k-вимірних наближень булевих функцій;

– створити пакет прикладних програм для оцінювання та обґрунтування стійкості синхронних потокових шифрів відносно запропонованих статистичних атак;

– обґрунтувати практичну стійкість шифру SNOW 2.0, що є прототипом майбутнього національного стандарту потокового шифрування, відносно узагальненої статистичної атаки;

– підвищити обґрунтованість експертних висновків про застосування в Україні перспективних алгоритмів потокового шифрування, призначених для захисту державних інформаційних ресурсів.

Наукові та практичні *результати дисертаційної роботи реалізовані* в Службі зовнішньої розвідки України – в результаті виконання НДР “Севрюга” (акт від 30.09.2016 р.) та “Мокрель” (акт від 30.09.2016 р.), а також в науково-технічних розробках ЗАО “Інститут інформаційних технологій” (акт від 25.07.2016 р.).

Особистий внесок здобувача. В статті [30] і тезах доповідей [36, 38 – 40, 42] автором запропоновано та реалізовано статистичну атаку на фільтрувальний генератор гами з лінійним законом реініціалізації початкового стану; в статті [34] і тезах доповідей [41, 43] автором запропоновано узагальнену статистичну атаку на синхронні потокові шифри; в статті [33] автором запропоновано та реалізовано швидкі алгоритми побудови списку k-вимірних наближень булевих функцій; в статті [31] – швидкий імовірнісний алгоритм оцінювання відстані між зрівноваженою булевою функцією та

множиною k-вимірних функцій; в статті [35] і тезах доповіді [44] розроблено та реалізовано алгоритми пошуку k-вимірних наближень булевих функцій та обчислення статистичних оцінок відносної відстані між вхідною функцією та множиною k-вимірних функцій.

Апробація результатів дисертації. Результати дисертаційних досліджень доповідалися та обговорювалися *на 8 міжнародних наукових та науково-практичних конференціях*: Міжнародному конгресі з інформатики "Информационные системы и технологии "CSISIT' 2013" (Білорусь, м. Мінськ, 2013 рік), XLI – XLII Міжнародних наукових конференціях "Питання оптимізації обчислень" (м. Кацівелі, 2013 р., смт. Чинадієво, 2015 р.), Міжнародній науковій конференції "Probability, reliability and stochastic optimization" (м. Київ, 2015 р.), XV – XVIII Міжнародних науково-практичних конференціях "Безпека інформації в інформаційно-телекомунікаційних системах" (м. Київ, 2012, 2013, 2015, 2016 рр.).

Публікації. Основні наукові результати дисертаційної роботи опубліковано в 14 наукових працях: з них 6 наукових статей [30 – 35] в наукових спеціалізованих виданнях України (2 видання індексуються міжнародними наукометричними базами); 8 тез доповідей на наукових та науково-практичних конференціях [36, 38 – 44].

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ ОЦІНЮВАННЯ ТА ОБҐРУНТУВАННЯ СТІЙКОСТІ ПОТОКОВИХ ШИФРІВ, ЩО ВИКОРИСТОВУЮТЬСЯ В СУЧАСНИХ ІНФОРМАЦІЙНО-ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМАХ

1.1. Роль та значення потокових шифрів у забезпеченні захисту інформації в сучасних інформаційно-телекомунікаційних системах

Забезпечення надійного захисту інформації в сучасних інформаційно-телекомунікаційних системах є складною проблемою, для вирішення якої застосовують організаційні, технічні та криптографічні методи. Останні базуються на математичних перетвореннях інформації, яка захищається з використанням допоміжних секретних даних (ключів), та є незамінними при вирішенні таких задач інформаційної безпеки як забезпечення конфіденційності, цілісності, автентичності, невідстежуваності (анонімності), неможливості відмови від авторства та ін. [3, 4]. Рівень розвитку криптографічних методів, що використовуються для захисту державних інформаційних ресурсів, здійснює суттєвий вплив на інформаційну безпеку держави, його економічну безпеку та суверенітет. Тому вдосконалення відомих та створення нових криптографічних методів та засобів, призначених для захисту інформації в інформаційно-телекомунікаційних системах України, є однією з пріоритетних задач науково-технічних організацій, спеціальних служб [45] та інших профільних (як державних так і комерційних) структур. Ця задача включає в себе, зокрема, розробку, вдосконалення та впровадження нових методів оцінювання та обґрунтування стійкості потокових шифрів, що використовуються на даний час як в Україні так і за її межами.

Особливості побудови та функціонування потокових шифрів обумовлюють їх широке використання в спеціальних інформаційно-телекомунікаційних системах з жорсткими вимогами до швидкодії (рис. 1.1), яку не можуть забезпечити блокові шифри (зокрема, програмні реалізації слово-орієнтованих потокових шифрів є в 5 – 10 разів швидше в порівнянні з відповідними реалізаціями блокових шифрів [5]). На сьогодні лише потокові шифри дозволяють забезпечити прийнятну швидкість шифрування на канальному, мережевому та транспортному рівнях. В останні роки спостерігається суттєве розширення сфери застосування потокових шифрів, що пов’язано з розвитком інформаційних технологій, засобів передачі даних та суттєвим прогресом у дослідженні криптографічних властивостей алгоритмів потокового шифрування [46 – 48].

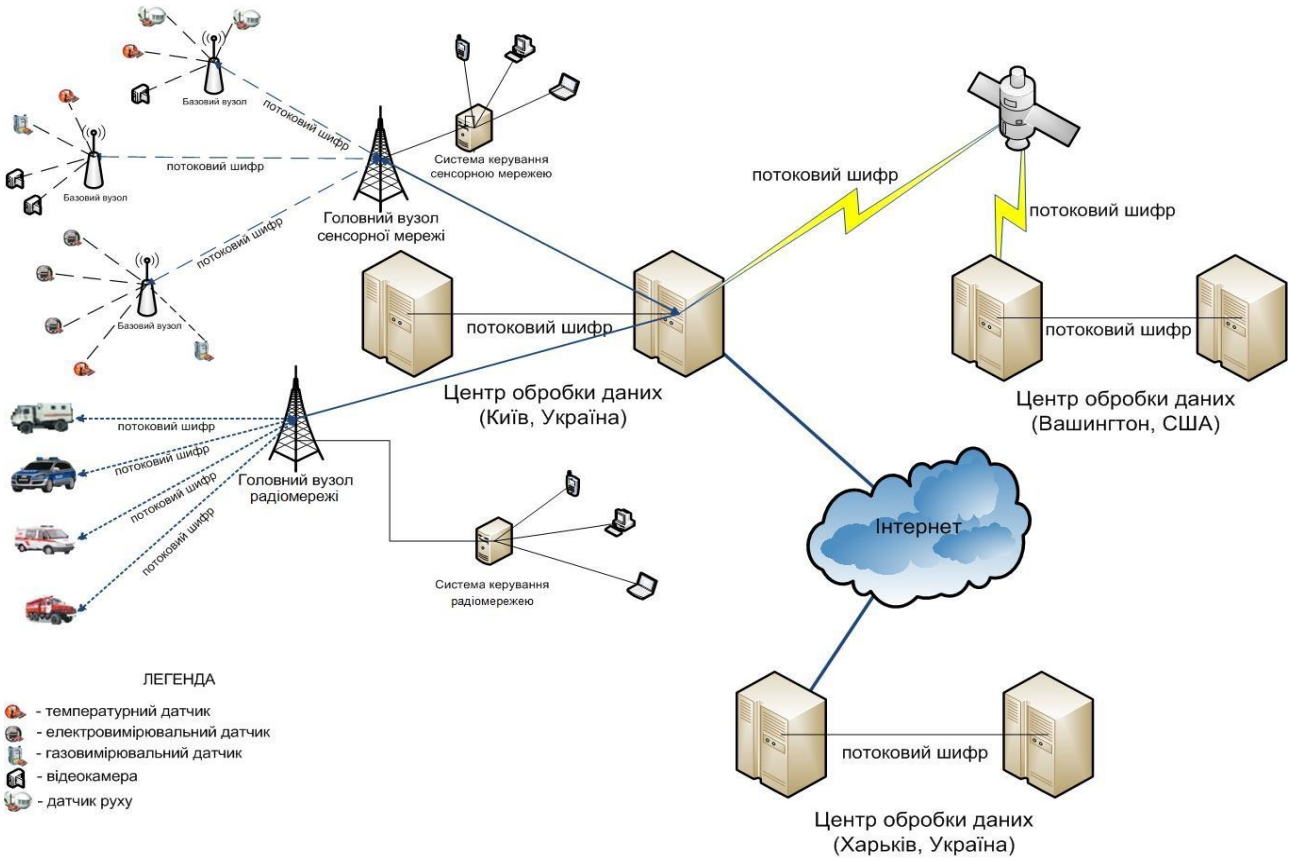

Рис. 1.1. Типова схема спеціальної інформаційно-телекомунікаційної системи

Програмно-орієнтовані потокові шифри знаходять застосування у вбудованих додатках систем з обмеженою кількістю обчислювальних ресурсів, зокрема, у бездротовій телефонії (шифр SOBER [6]), в системах комутативного зв'язку (Bluetooth, Wi-Fi та інш. [49 – 51]), в системах платного телебачення (шифри Panama [7], Chameleon [52]). Окремо варто відзначити алгоритм потокового шифрування A5/1, який є невід'ємною частиною стандарту мобільного зв'язку GSM [8, 50].

Останнім часом значний розвиток отримали бездротові сенсорні мережі (БСМ). Стандартна БСМ (рис. 1.2) складається з декількох мініатюрних вузлів, оснащених малопотужним приймально-передавальним пристроєм, мікропроцесором та сенсором. Сукупність таких розподілених БСМ може пов'язувати в єдину систему глобальні комп'ютерні мережі та локальні фізичні середовища. Концепція бездротових сенсорних мереж привертає увагу багатьох вчених, дослідних інститутів та комерційних організацій, що забезпечує великий обсяг наукових робіт з даної тематики [53, 146 – 155]. Значний інтерес до БСМ обумовлений широкими можливостями їхнього застосування.

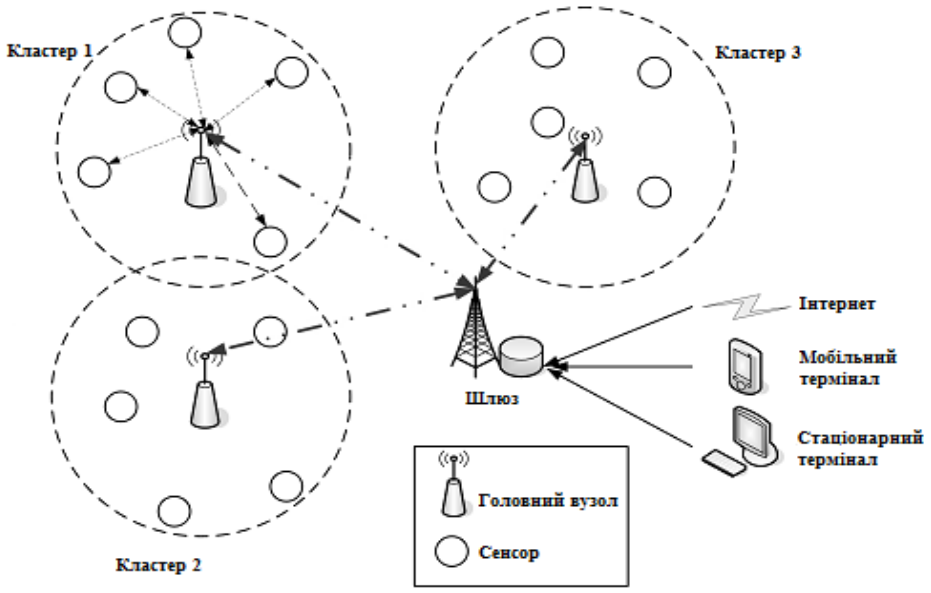

Рис. 1.2 Схема бездротової сенсорної мережі

Зокрема, БСМ, можуть використовуватися для передбачення відмови обладнання авіаційних та аерокосмічних систем (рис. 1.3), а також автоматизації промислових будівель [54]. Через здатність до самоорганізації, автономності та високої відмовостійкості БСМ активно використовуються в системах безпеки та військових додатках [55], в галузі моніторингу довкілля [56, 57] та в медичній галузі [58].

Бездротова технологія та автономність БСМ породжує нові загрози та ризики до безпеки інформації, що циркулює всередині мережі. Розміщення компонентів БСМ поза контрольованою зоною робить їх вразливими до фізичного знищення або несанкціонованої модифікації [59]. Крім того, оскільки в БСМ має місце передача даних за допомогою радіохвиль, то можливе проведення широкого класу атак, починаючи з пасивного прослуховування, спрямованого на аналіз трафіку окремого вузла або всієї БСМ, та закінчуючи найбільш небезпечними активними атаками [60 – 63], які можуть визвати перезавантаження/відмову вузлів мережі чи нав’язати передачу хибної інформації.

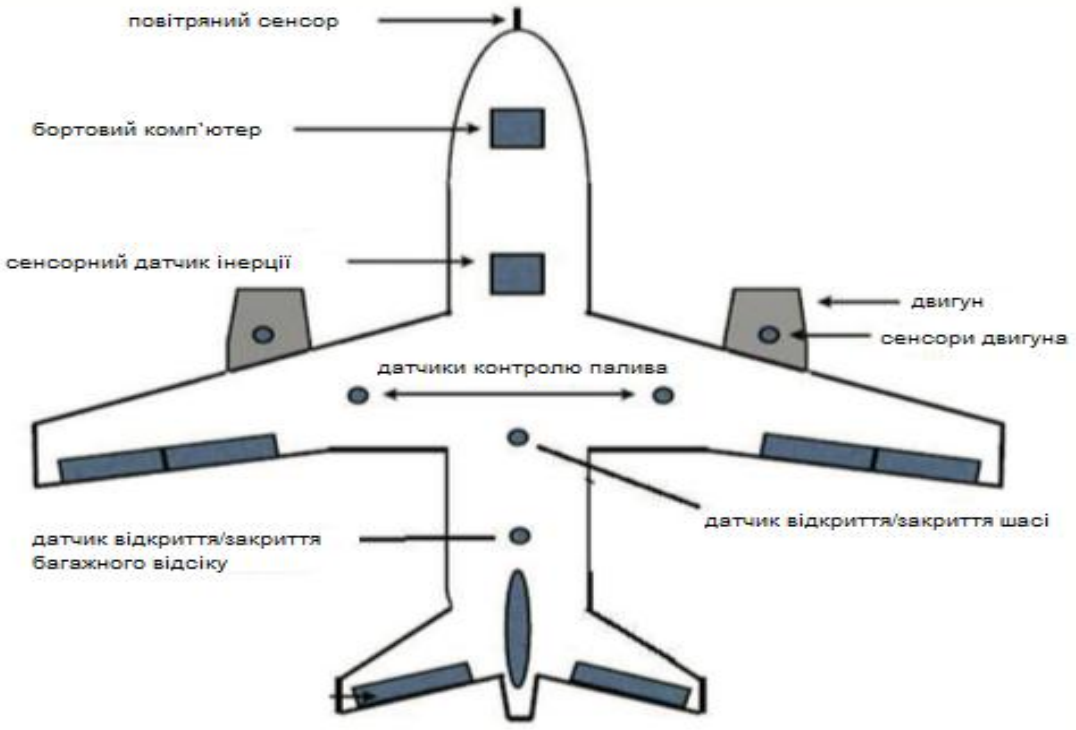

Рис. 1.3. Бездротова сенсорна мережа літаку

Для захисту інформації в БСМ, поряд з блоковими, використовуються і потокові шифри, оскільки їх застосування не призводить до розмноження помилок [64]. Крім того, програмна реалізація потокового шифру зазвичай потребує значно менше пам’яті ніж відповідна реалізація блокового шифру.

Варто відзначити ще одну галузь застосування програмно-орієнтованих потокових шифрів, а саме, в супутникових системах зв’язку. Супутниковий зв'язок – це один з різновидів космічного радіозв’язку, який базується на використанні штучних супутників Землі в якості ретрансляторів. Супутниковий зв'язок здійснюється між земними станціями, які можуть бути як стаціонарними так і рухомими. Супутниковий зв'язок є розвитком традиційного радіорелейного зв’язку шляхом виносу ретранслятору на дуже велику висоту. Оскільки зона його видимості в цьому випадку – майже половина Земної кулі, то необхідність в ланцюгу ретрансляторів відпадає: в більшості випадків вистачає і одного.

Основними перевагами супутникових систем зв’язку є:

1) велика пропускна здатність, обумовлена роботою супутників в широкому діапазоні частот;

2) забезпечення зв’язком на величезних відстанях та можливість обслуговування абонентів в самих важкодоступних місцях;

3) можливість побудови мережі зв’язку без фізично реалізованих комутаційних пристроїв, обумовлена широкомовністю роботи супутників.

До недоліків відносяться:

1) найбільша вартість зв’язку в порівнянні з іншими телекомунікаційними системами;

2) наявність затримки приймання радіосигналу наземною станцією через великі відстані до супутника;

3) схильність супутникових сигналів до впливу різноманітних атмосферних явищ.

В роботах [65, 66] наведено опис реверсивного інженерингу та успішні атаки на алгоритми потокового шифрування, які використовуються в телекомунікаційних стандартах GMR-1 та GMR-2, що реалізуються в переважній більшості сучасних систем супутникового зв’язку (Thuraya, SkyTerra, TerreStar, Inmarsat). Показано, що за допомогою стандартного ПК та звичайного радіоприймача можна прослуховувати абонентів, які використовують супутниковий телефонний зв'язок. Зазначена робота являє собою наочний приклад ненадійності шифрів, які не висувалися для публічного огляду та аналізу, а головним критерієм “стійкості” яких є секретність їх описів (*security through obscurity*).

Таким чином, впровадження БСМ та/або супутникових систем зв’язку для роботи в державних установах та організаціях загальнонаціонального значення створює певні загрози інформаційної безпеки, тому конфіденційність інформації в таких системах повинна забезпечуватися стандартизованими алгоритмами шифрування, що мають науково обґрунтовану стійкість.

Значна увага до потокових шифрів приділяється також у науковому криптографічному співтоваристві Європейського союзу. Про це свідчить проведення міжнародних конкурсів NESSIE (2000 – 2003 рр.) та eSTREAM (2004 – 2008 рр.), спрямованих на створення стійких потокових шифрів, придатних до широкого застосування. Процес стандартизації потокових шифрів в західних країнах триває вже більше 10 років. На сьогодні стандартизованими в ISO/IEC є такі потокові алгоритми шифрування: A5/1, A5/2, A5/3 [8, 50], RC4 [67], SNOW-2.0 (рис. 1.4) [5], Enocoro [68], Trivium [69], MUGI [70], Rabbit [71], Decim-v2 [72], KCipher-2 (K2) [73]. Зауважимо, що алгоритми сім’ї A5 та шифр RC4 визнано слабкими, в той час як на інші з перелічених алгоритмів не виявлено ефективних атак, і вони, в принципі, придатні до широкого застосування.

Через відсутність на даний час національного стандарту потокового шифрування в Україні вирішення цієї задачі є дуже актуальним для нашої держави.

1.2. Особливості побудови та функціонування сучасних синхронних потокових шифрів

На сьогодні провідне положення в галузі систем потокового шифрування, що використовуються для забезпечення конфіденційності інформації в спеціальних інформаційно-телекомунікаційних системах України та інших країн, належить синхронним потоковим шифрам (СПШ) [74, 75, 64, 48].

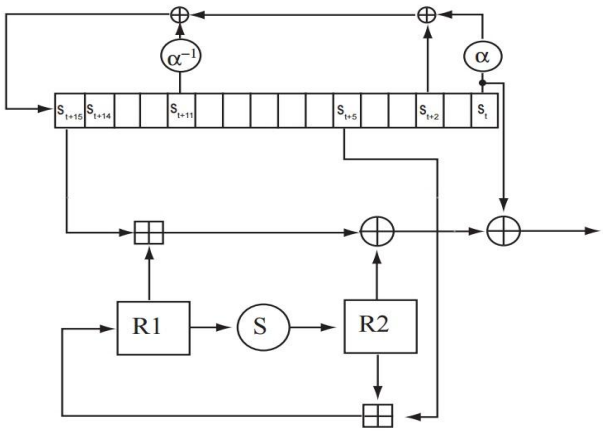

Рис. 1.4. Схема генератора гами шифру SNOW-2.0

Неформально *синхронний потоковий шифр* визначається як такий, що реалізує зашифровування з використанням криптографічних перетворень, які не залежать від відкритих або шифрованих повідомлень [64, 74, 75]. До переваг СПШ відносяться можливість компактної апаратної реалізації засобів шифрування (з невеликою схемною складністю), забезпечення високої швидкості зашифровування-розшифровування та відсутність ефекту

розповсюдження помилок [64, 74, 75]. Отже, застосування СПШ є доцільним саме у тих випадках, коли символи повідомлень повинні оброблятися індивідуально, в порядку їхнього отримання, а ймовірність спотворень при передачі є достатньо високою. Зауважимо, що асинхронні потокові шифри характеризуються властивістю обмеженого розповсюдження помилок. Більшість з них будується на основі блокових шифрів, що застосовуються в режимі CFB [75].

Найсуттєвішою особливістю практичного застосування СПШ є необхідність здійснення процедури синхронізації шифраторів передачі та прийому інформації [64, 74]. Зазначені шифратори повинні функціонувати строго узгоджено, оскільки розшифровування повідомлень не може бути здійснено вірно, поки не синхронізовані шифрувальні гамми, що виробляються на передавальному та приймальному кінцях лінії зв’язку. Звичайно відновлення синхронізації вимагає пошуку можливих зсувів між тактами роботи пристроїв відправника та одержувача. Один з можливих способів такого відновлення полягає у використанні спеціальних “маркерів”, що вставляються у шифроване повідомлення. В результаті будь-який символ шифротексту, що пропущено при передачі, призводить до невірного розшифровування лише доти, поки не буде прийнято один з “маркерів”.

Інше відоме рішення проблеми синхронізації СПШ полягає в одночасній реініціалізації станів шифраторів передачі та прийому інформації, що здійснюється за деякою попередньо узгодженою умовою [64, 74]. Під час реініціалізації початкового стану шифратора (генератора гами СПШ) ключова інформація “перемішується” певним чином з вектором ініціалізації (синхропосилкою), що передається відкритим каналом зв’язку. Це забезпечує неможливість повторного використання однієї і тієї ж гами, тобто унеможливлює так зване перекриття шифру [64, 76].

Здійснення синхронізації шифрувальних пристроїв є необхідною умовою надійного функціонування СПШ, яка тягне за собою можливість проведення на

них специфічних атак на основі відомих або підібраних векторів ініціалізації [9, 10, 25 – 28]. Стійкість відносно даних атак є обов'язковою вимогою до сучасних СПШ.

Переважна більшість сучасних синхронних потокових шифрів являє собою адитивні шифри (модульного або імпульсного гамування), що будуються на основі *генераторів псевдовипадкових послідовностей* (*гам*). Відповідно до сучасного уявлення [77, 78], будь-який (адитивний) *синхронний потоковий шифр* визначається як сукупність двох алгоритмів: 1) генерації шифрувальної гами; 2) формування початкового стану (ПС) генератора гами (ГГ) за ключем шифрування та вектором ініціалізації. Як правило, останній алгоритм використовується також для реініціалізації ПС при фіксованій або вимушеній синхронізації шифраторів передачі та прийому інформації [28, 74].

Стандартною математичною моделлю генератора гами СПШ є скінченний автономний автомат $\Im = (S, Y, h, f)$, де S та Y позначають внутрішній і вихідний алфавіти автомату $\Im$ відповідно, а $h: S \to S$ і $f: S \to Y$ є, відповідно, функціями переходів і виходів даного автомату (див., наприклад, [74]). Звичайно множина Y складається з двох елементів: $Y = \{0, 1\}$, а множина S співпадає із сукупністю V_N усіх булевих векторів певної довжини N.

На рис. 1.5 зображено процедури зашифровування та розшифровування двійкових повідомлень з використанням шифру імпульсного гамування з генератором гами $\Im$. Спочатку за даним ключем k та вектором ініціалізації c у відповідності з певним алгоритмом H обчислюється початковий стан $x(0)$ автомату $\Im$:

$$x(0) = H(k, c), \tag{1.1}$$

за яким далі формуються його внутрішня послідовність $\bar{x} = \{x(i) : i = 0,1,...\}$ та шифрувальна гама $\bar{\gamma} = \{\gamma_i : i = 0,1,...\}$, де

$$x(i+1) = h(x(i)),\ \gamma_i = f(x(i)),\ i = 0,1,\ldots. \qquad (1.2)$$

Символи t_i та s_i відкритого та, відповідно, шифрованого повідомлень у i-му такті шифрування пов'язані співвідношенням $s_i = t_i \oplus \gamma_i$, $i = 0, 1, \ldots$, де $\oplus$ позначає операцію додавання за модулем 2.

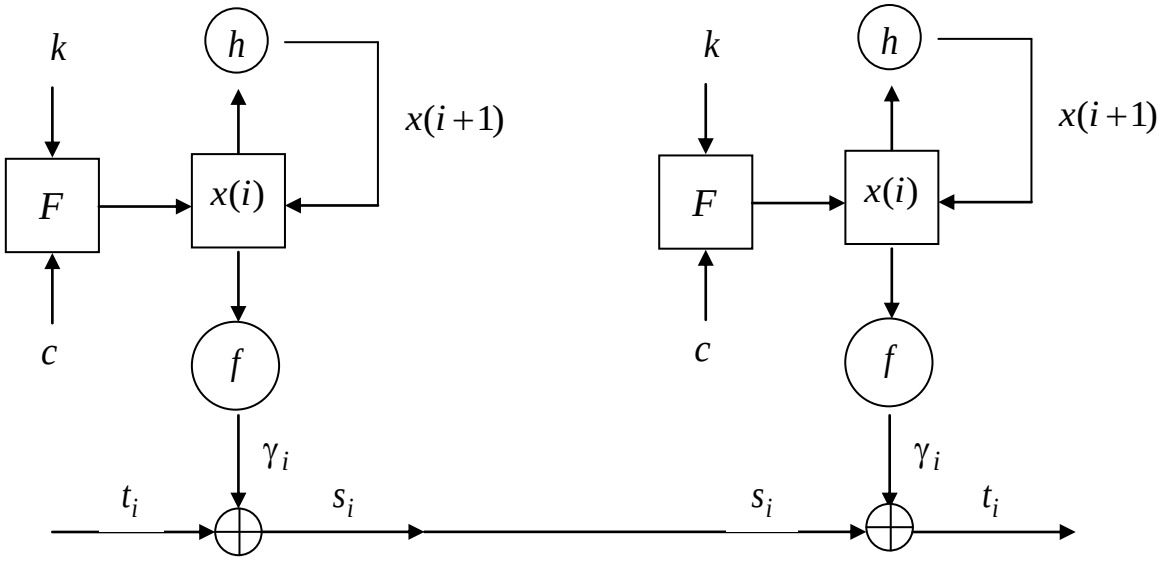

Рис. 1.5. Схематичне зображення процедури шифрування повідомлень з використанням синхронного шифру імпульсного гамування

Іноді алгоритм формування початкового стану $x(0)$ за даними ключем k та вектором ініціалізації c реалізується з використанням певного лінійного перетворення $L: V_{l_0} \times V_{l_1} \to V_N$ над полем з двох елементів:

$$H(k,c) = L(k,c),\ k \in V_{l_0}, c \in V_{l_1},$$

що обумовлено, головним чином, вимогами простоти реалізації та високої швидкості виконання процедури реініціалізації ПС. Проте, як показано у [28],

застосування лінійних двійкових перетворень при формуванні початкового стану генератора гами СПШ знижує стійкість останнього відносно багатьох (зокрема, алгебраїчних) атак.

Традиційний метод синтезу ГГ полягає в побудові генератора у вигляді послідовного з'єднання двох автоматів: генератора попередньої гами та блоку ускладнення. В ролі генераторів попередніх гам, як правило, використовуються лінійні регістри зсуву (ЛРЗ), які, за умовою примітивності їх многочленів зворотного зв'язку, дозволяють отримувати псевдовипадкові послідовності з гарними статистичними та структурними властивостями [74]. Призначення блоку ускладнення полягає у запобіганні простоти аналітичної (лінійної) залежності, що пов'язує знаки вихідної послідовності ЛРЗ з його початковим станом, тобто, фактично, з ключем шифрування.

Найважливішою вимогою до потокових шифрів є умова їх практичної стійкості відносно всіх відомих криптоаналітичних атак. Якщо ця вимога не виконується, то інші характеристики шифру втрачають своє значення. В той же час, отримання науково обґрунтованих оцінок стійкості потокових шифрів (навіть відносно конкретних, добре вивчених методів криптоаналізу) є складною науковою проблемою. Фактично висновок про стійкість будь-якого потокового шифру ґрунтується на неможливості провести на нього окремі атаки, відомі криптоаналітикам, а також на припущенні про те, що майбутні атаки не призведуть до помітних покращень відомих криптоаналітичних методів. У зв'язку з цим особливу гостроту набуває проблема обґрунтування стійкості потокових шифрів, призначених для захисту інформації в спеціальних інформаційно-телекомунікаційних системах України. Частиною цієї проблеми є наукова задача даної дисертаційної роботи, яка полягає в розробці методу побудови науково обґрунтованих оцінок стійкості потокових шифрів відносно найбільш потужних (серед відомих на сьогодні) статистичних атак на основі підібраних векторів ініціалізації.

1.3. Аналіз методів оцінювання та обґрунтування стійкості потокових шифрів відносно сучасних атак

1.3.1. Загальний огляд методів криптоаналізу синхронних потокових шифрів.

Як відомо [79], криптографічний аналіз являє собою напрям криптології, що вивчає основні закономірності, протиріччя, методи та засоби аналізу криптографічних систем, які можуть бути використані для порушення конфіденційності, цілісності, справжності, доступності та спостережливості інформації та ресурсів. Головним предметом дослідження у криптографічному аналізі є стійкість криптосистем, під якою розуміють їхню спроможність протистояти усім відомим криптоаналітичним атакам [64, 74]. Під методом криптографічного аналізу розуміють сукупність прийомів та способів, що спрямовані на оцінювання стійкості криптосистеми та поєднані однією чи декількома спільними ідеями. Будь-який алгоритм, спрямований на зниження рівня безпеки криптосистеми на основі певного методу криптоаналізу, називається атакою на криптосистему. Отже, атаки є фактично реалізаціями методів криптоаналізу [79].

Сучасні методи криптоаналізу синхронних потокових шифрів (а також криптоаналітичні атаки, що будуються на їх основі) можна розділити на методи “зламування”, які спрямовані на відновлення ключів або відкритих повідомлень, та методи, метою яких є виявлення відмінностей між вихідними послідовностями генератора гами потокового шифру та суто випадковими послідовностями (рис. 1.6). На основі останніх методів будуються так звані *розрізнювальні атаки* на потокові шифри, в розробці яких за останні роки досягнуто значний прогрес [80 – 84]. В залежності від інформації, яка доступна криптоаналітику, та його можливостей, криптоаналітичні атаки прийнято поділяти на такі класи [3, 75]:

- атаки на основі відомого шифрованого тексту;
- атаки на основі відомого відкритого тексту;
- атаки на основі підібраних відкритих текстів;
- атаки на основі відомих чи підібраних векторів ініціалізації.

Останній тип атак є специфічним для синхронних потокових шифрів, хоча за сутністю вони не відрізняються від атак зі зв'язаними ключами на блокові шифри [85]. Можливість практичного проведення атаки на основі відомих векторів ініціалізації виникає за умови здійснення синхронізації шифрувальної апаратури у фіксовані дискретні моменти часу (як приклади потокових шифрів з "фіксованим механізмом ресинхронізації", відзначимо СПШ E0 [51] та A5/1 [8, 50]). Атака на основі підібраних векторів ініціалізації може бути проведена у тому випадку, коли при втраті синхронізації відбувається вимушена ресинхронізація шифраторів передачі та прийому інформації [28, 74]. Стійкість відносно подібних атак є стандартною вимогою до сучасних СПШ [28, 47].

Іншою ознакою, за якою класифікують криптоаналітичні атаки, є математичний апарат, що використовується при розв'язанні відповідної задачі криптоаналізу. Умовно методи криптоаналізу СПШ можна розділити на *методи опробування ключів*, *алгебраїчні* та *статистичні методи* (див. рис. 1.6.). Основними показниками ефективності зазначених методів є *трудомісткість* (або *часова складність*), що визначається як кількість тих чи інших операцій, необхідних для розв'язання задачі криптоаналізу, та *надійність* (або *достовірність*), тобто ймовірність правильного відновлення ключа шифрування з використанням даного методу [76, 86].

Оскільки надійність методу криптоаналізу як правило, залежить від обсягу доступного шифроматеріалу, то для характеристики ефективності методу часто використовують ще один показник, що визначається як найменший обсяг матеріалу, достатній для відновлення ключа з потрібною надійністю [76].

Найбільш відомим, універсальним методом криптоаналізу є метод *тотального опробування ключів*, практичне застосування якого може бути успішним лише у випадку невеликої довжини ключа [74, 76]. Для скорочення часових витрат при опробуванні використовуються *методи групування* та *методи упорядкування ключів* [74, 87], в основі яких лежать, відповідно, ідеї відбраковування ключа в цілому при опробуванні його частини та проведення

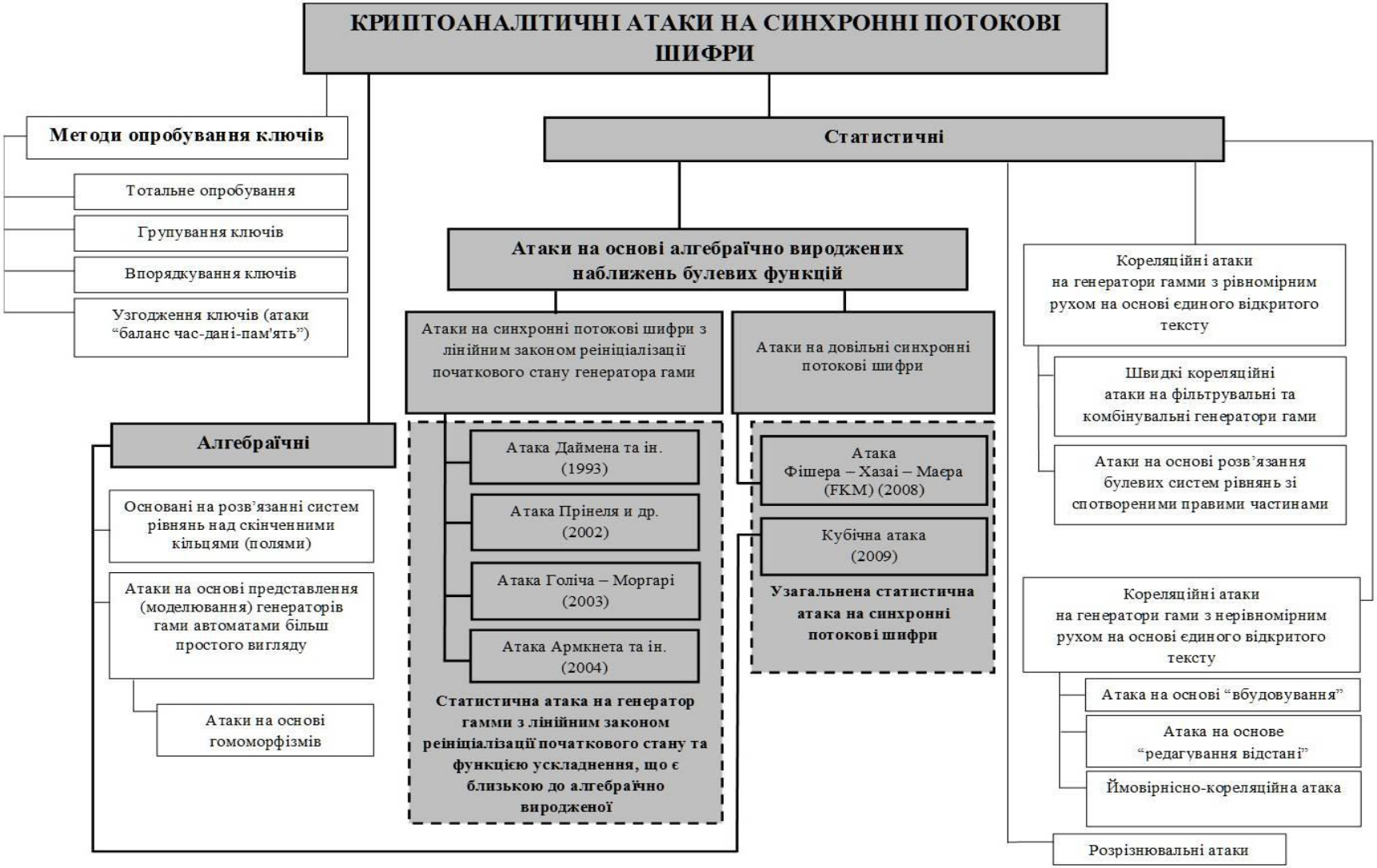

Рис. 1.6. Криптоаналітичні атаки на синхронні потокові шифри

опробування, починаючи з найбільш ймовірних ключів (при нерівномірному розподілі ймовірностей на множині ключів шифру).

Одним з потужних загальних підходів до криптоаналізу потокових шифрів є застосування групи методів та алгоритмів, що отримали назву *“баланс час-дані-пам’ять”* (time memory trade-off) [88 – 93]. Перша атака такого типу на потокові шифри запропонована в [90] та застосована пізніше для криптоаналізу шифру A5/1 [88, 91]. Узагальнення відомих атак “баланс час-дані-пам’ять” і детальне дослідження їх ефективності та умов застосовності проведено в [92, 93]. Зокрема, у [92] показано, що для практичної стійкості потокового шифру відносно таких атак довжина початкового стану його генератора гами повинна бути, як мінімум, у двічі більше за довжину ключа, яка, у свою чергу, повинна дорівнювати довжині вектора ініціалізації.

На практиці більш ефективними у порівнянні з методами опробування ключів є *алгебраїчні атаки* на ПШ, що полягають у відновленні невідомого ключа шляхом розв’язання деякої системи рівнянь [74, 76, 94, 95]. Зауважимо, що при аналізі реальних потокових шифрів їхні системи рівнянь гамоутворення можуть бути відомі не повністю, а у ряді випадків навіть складання таких систем є неможливим, внаслідок складності криптосхем СПШ. Крім того, для складеної системи рівнянь повинні бути розроблені ефективні алгоритми розв’язання. Отже, практичне застосування алгебраїчних атак приводить до відновлення ключа за виконанням таких умов [74, 76]:

1) для заданого шифру (або його генератора гами) може бути складена система рівнянь відносно невідомих ключових параметрів;

2) вихідні дані, за якими складається система рівнянь, не повинні містити спотворень; у противному випадку невідомі параметри можуть бути знайдені помилково у випадку сумісності системи, або система в цілому виявиться несумісною;

3) повинен існувати алгоритм розв’язання отриманої системи рівнянь із прийнятною обчислювальною складністю.

Алгебраїчні методи криптоаналізу комбінувальних та фільтрувальних генераторів гами (з рівномірним рухом ЛРЗ) отримали значний розвиток у роботах [94, 96 – 105], де запропоновано низку так званих швидких алгебраїчних атак на такі генератори. Сутність цих атак полягає у побудові системи нелінійних алгебраїчних рівнянь відносно невисокого степеня, яка пов'язує ПС генератора гами з його вихідною послідовністю, та подальшому розв'язанні цієї системи рівнянь методом лінеаризації або іншими відомими методами [95, 106, 107]. З використанням швидких алгебраїчних атак були розроблені більш ефективні, у порівнянні з повним перебором ключів, алгоритми криптоаналізу шифрів Toyocrypt, LILI-128 [96] та шифру E0 [98, 99]. На сьогодні теорія побудови зазначених атак знаходиться на етапі інтенсивного розвитку та відокремлюється у відносно самостійний напрям криптоаналізу потокових шифрів.

З практичного погляду, найбільш ефективними є *статистичні методи криптоаналізу* потокових шифрів, сутність яких полягає у розв'язанні задачі пошуку невідомих ключових параметрів з використанням процедур статистичної класифікації [86]. Кожна така процедура визначає розбиття простору спостережень (шифрованих повідомлень або вихідних послідовностей генератора гами СПШ) на певні області, що не перетинаються. Якщо апостеріорні ймовірності ключа шифрування (або його частини) за умови потрапляння спостереження в зазначені області відрізняються між собою, то при достатньо великій кількості спостережень можна з визначеною достовірністю розрізнити, за яким законом розподілені дані спостереження, а, отже, і відновити шуканий ключовий параметр. Як правило, застосування статистичних методів криптоаналізу є можливим за менш жорстких обмежень у порівнянні з алгебраїчними методами, що й обумовлює їх широке використання на практиці та високу результативність.

Найбільш розвинутий клас статистичних атак на генератори гами СПШ складають так звані *кореляційні атаки*, які базуються на статистичній

залежності між внутрішніми станами генератора та його вихідними послідовностями [74, 75, 86]. Першою відкритою роботою, де запропоновано кореляційну атаку на комбінувальний генератор гами, є стаття [108]. Пізніше у роботах [109 – 113] запропоновано низку швидких кореляційних атак на фільтрувальні та комбінувальні генератори гами з рівномірним рухом, які дозволяють за певних умов достатньо надійно відновлювати ПС зазначених генераторів із поліноміальною (від довжини ПС) складністю. На сьогодні кореляційним атакам на СПШ присвячені десятки наукових робіт [108 – 127]. Найбільш ефективними вважаються кореляційні атаки, що базуються на методах декодування згорточних кодів [122, 124, 125], турбокодів [123] та блокових кодів з малою щільністю перевірок на парність [121]. Такі атаки дозволяють практично зламувати генератори гами, що будуються на основі ЛРЗ довжиною до 100 біт, за умови, що ймовірність спотворення знаків відповідної лінійної рекурентної послідовності не надто близька до 0,5 [48]. Іншим важливим досягненням є кореляційні атаки високого порядку, які базуються на статистичній близькості заданої нелінійної функції до множини булевих функцій малого степеня [96, 128, 129], а також кореляційні атаки на генератори гами з пам’яттю [130].

На сьогодні методи криптоаналізу потокових шифрів та, зокрема, статистичні атаки на СПШ знаходяться на етапі інтенсивного розвитку. Проте однією з центральних задач подальших досліджень в цій галузі є систематизація та уніфікація відомих атак з метою створення загальних методів оцінювання та обгрунтування стійкості СПШ відносно них.

1.3.2. Атаки на синхронні потокові шифри на основі алгебраїчно вироджених наближень булевих функцій.

Проведений аналіз наукових публікацій показує, що переважна більшість найпотужніших статистичних атак на СПШ будується за допомогою певних наближень (апроксимацій) булевих функцій, пов’язаних з алгоритмами

шифрування, функціями менш складної структури. До останніх відносяться афінні (зокрема, лінійні) функції, а також функції від малої кількості змінних.

Загальний клас булевих функцій, який включає в себе зазначені окремі класи, утворюють алгебраїчно вироджені функції.

За означенням [23, 24] булева функція $g: V_n \to \{0,1\}$ називається *k-вимірною,* $0 \le k \le n$, якщо існує функція $\varphi: V_k \to \{0,1\}$ та $n \times k$-матриця A над полем з двох елементів такі, що для будь-якого $x \in V_n$ виконується рівність $g(x) = \varphi(xA)$. Функція g називається *алгебраїчно виродженою*, якщо вона є k-вимірною для деякого $k < n$, та *невиродженою* – в протилежному випадку [29, 131, 132].

Позначимо $B_{n,k}$ множину всіх k-вимірних функцій від n змінних. Справедливі співвідношення $B_{n,0} \subseteq B_{n,1} \subseteq \ldots \subseteq B_{n,n-1} \subseteq B_{n,n}$; при цьому множина $B_{n,0}$ складається з двох функцій-констант, множина $B_{n,1}$ співпадає з класом афінних функцій, а множина $B_{n,n}$ – з сукупністю всіх булевих функцій від n змінних. Функції з множини $B_{n,n-1}$ є алгебраїчно виродженими, а функції з множини $B_n \setminus B_{n,n-1}$ – невиродженими.

Відносна відстань між функціями $f: V_n \to \{0,1\}$ та $g: V_n \to \{0,1\}$ визначається за формулою $d(f,g) = 2^{-n} |\{x \in V_n : f(x) \ne g(x)\}|$. При цьому функція g називається *наближенням* (або *ε-наближенням,* $\varepsilon \in (0, 1/2)$) функції f, якщо виконується умова $d(f,g) < \varepsilon$.

Атаки на СПШ, що базуються на алгебраїчно вироджених наближеннях булевих функцій, можна поділити на два класи. До першого з них відносяться атаки на ГГ з лінійним законом реініціалізації ПС, а до другого – атаки на довільні СПШ (рис. 1.6). Одна з перших таких атак запропонована в [25], де показано, що якщо закон реініціалізації ПС є лінійним, а функція ускладнення ГГ залежить від $s < l_0$ змінних, то на СПШ можна провести алгебраїчну атаку

на основі підібраних векторів ініціалізації, складність якої є $O(tsl_0^2 + t2^s)$ операцій, де t позначає довжину відрізку гами, який виробляється генератором при кожному запуску (новому значенні вектора ініціалізації), l_0 є довжиною ключа; при цьому потрібно, щоб кількість запусків була не менше, ніж 2^s.

Атака з [25] вдосконалювалася та узагальнювалася в різних напрямах [26 – 28]. В [27] показано, що вона є застосовною до генератора гами з алгебраїчно виродженою функцією ускладнення порядку $n-s$ (тобто функцією від $n>s$ змінних, що є лінійно еквівалентною до функції від s змінних); розглянуто також випадок, в якому зазначена функція може бути невідомою криптоаналітику. В [28], поряд з іншими, описано статистичну атаку на генератор гами з лінійним законом реініціалізації ПС та довільною функцією ускладнення, що є близькою до афінної булевої функції. Відзначимо також роботу [37], в якій запропоновано алгоритм відновлення початкового стану ГГ за єдиним відрізком вихідної послідовності в припущені, що криптоаналітик має доступ до знаків цієї послідовності у визначених тактах, номера яких експоненційно залежать від довжини початкового стану ГГ.

На сьогодні серед найбільш потужних атак, що базуються на алгебраїчно вироджених наближеннях булевих функцій, слід відзначити кубічну атаку [9], статистичну атаку Фішера-Хазаі-Майєра (FKM) [10], а також їх різноманітні модифікації та вдосконалення [11 – 16] (рис. 1.6). В принципі, подібні атаки можуть бути застосовані до будь-якого криптографічного алгоритму, який описується за допомогою булевої функції $F: V_{l_0} \times V_{l_1} \to \{0,1\}$, один з аргументів якої є секретним, а інший – загальнодоступним параметром. В ролі такої функції можна використовувати знак вихідної послідовності (який розглядається як функція ключа $k \in V_{l_0}$ та вектора ініціалізації $c \in V_{l_1}$) генератора гами в деякому такті. Припускається, що функція F доступна противнику у вигляді оракула (“чорної скрині”), зокрема, алгоритм, який реалізує цю функцію, може бути невідомим.

На етапі передобчислень противник може подавати на вхід оракула будь-які пари векторів $(x, y) \in V_{l_0} \times V_{l_1}$, обчислюючи значення $F(x, y)$, щоб зібрати необхідну інформацію про властивості функції F. Після цього противник отримує доступ до оракула $F_k(c) = F(k, c)$, $c \in V_{l_1}$, де значення ключа $k \in V_{l_0}$ є невідомим. Противник може вибирати будь-які вектори $c \in V_{l_1}$ та обчислювати значення $F_k(c)$ при фіксованому ключі k, намагаючись відновити цей ключ (або отримати про нього деяку інформацію). Іншою можливою стратегією противника є побудова розрізнювальної атаки, яка спрямована на те, щоб статистично розрізнити (за прийнятний час з достатньо високою надійністю) відображення F_k від випадкового рівноймовірного відображення $\Phi : V_{l_1} \to \{0, 1\}$ [11, 12].

Атака FKM [10] будується на основі наближення функції F булевою функцією g, яка залежить лише від деяких розрядів ключа. Це дозволяє спочатку відновити зазначені розряди методом максимуму правдоподібності, а потім знайти частину ключа, яка залишилася, шляхом повного перебору. В [10] на прикладах СПШ Grain-128 (рис. 1.7) та Trivium наведено способи вибору функцій F та g для побудови атаки; зокрема, показано, що можна побудувати атаку на редуковану версію шифру Grain-128, складність якої дорівнює 2^{124} операцій.

Поряд з тим, в [10] (а також інших доступних публікаціях) відсутні загальні методи оцінювання стійкості СПШ відносно подібних атак. Для розробки таких методів необхідно уніфікувати та узагальнити найбільш потужні на сьогодні атаки зазначеного типу, точно сформулювати загальні умови їх застосовності та отримати аналітичні оцінки параметрів, що характеризують їх ефективність (трудомісткості та обсягу матеріалу, необхідного для успішної реалізації атаки з заданою надійністю).

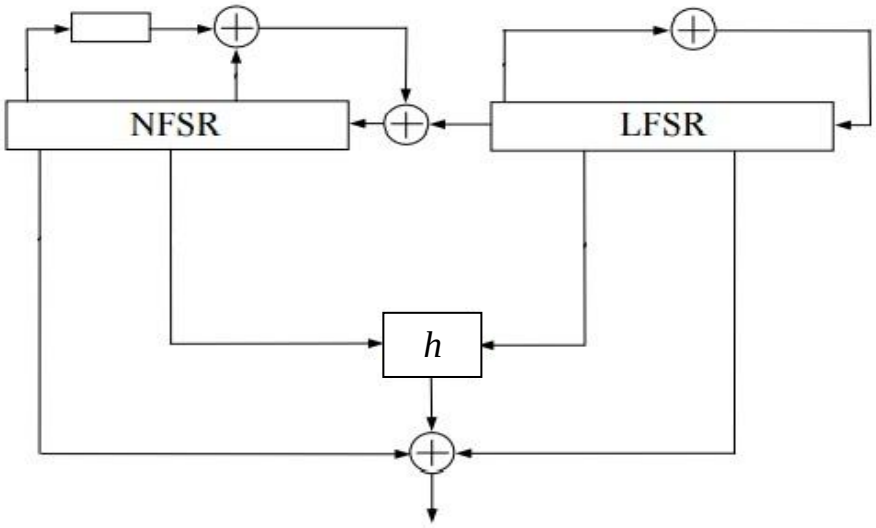

Рис. 1.7. Схема генератора гами шифру Grain-128

1.3.3. Огляд відомих методів побудови та аналізу алгебраїчно вироджених наближень булевих функцій. Як зазначено вище, переважна більшість найпотужніших статистичних атак на СПШ будується за допомогою алгебраїчно вироджених наближень булевих функцій, пов'язаних з алгоритмами шифрування. Для оцінювання та обґрунтування стійкості СПШ відносно таких атак необхідно будувати зазначені наближення або переконуватися в їх відсутності, зокрема, оцінювати відносну відстань між заданою булевою функцією та множиною шуканих наближень. Перелічені задачі є обчислювально складними, а ефективні алгоритми їх розв'язання відомі лише для окремих випадків [10, 17 – 24].

Найвідомішою є задача побудови лінійних наближень булевих функцій, що полягає в розробці алгоритмів, які формують для довільних $f : V_n \to \{0,1\}$ та $\varepsilon \in (0,1)$ список усіх лінійних функцій, що співпадають з функцією f не менше ніж на $2^{n-1}(1+\varepsilon)$ двійкових наборах. Класичним алгоритмом розв'язання цієї задачі є алгоритм швидкого перетворення Адамара, який обчислює усі лінійні наближення булевої функції від n змінних за $O(n2^n)$

операцій додавання та віднімання цілих чисел (див., наприклад, [86], с. 218). Зазначений алгоритм не є оптимальним за складністю, навіть у класі детермінованих алгоритмів [133].

Починаючи з публікації О. Гольдрайха та Л. Левіна [17], відомо низку ймовірнісних алгоритмів [18 – 22, 134 – 136], які дозволяють формувати список лінійних наближень довільної булевої функції від n змінних за поліноміальний від n час. Відзначимо серед них алгоритм Тревісана [21] зі складністю $O(n\varepsilon^{-4}\log n)$ операцій над n-розрядними цілими числами, вдосконалений алгоритм Левіна [134, 135], часова складність якого є $O(n\varepsilon^{-2}\log n\log(\varepsilon^{-2}))$ тих самих операцій та швидкий алгоритм Гольдрайха-Левіна [136], який має таку ж саму складність.

Зауважимо, що алгоритми, викладені у [17 – 22, 134 – 136], призначені саме для побудови всіх лінійних наближень, що знаходяться від заданої булевої функції на відстані, яка не перевищує певну межу. Поряд з тим, в окремих задачах криптоаналізу, наприклад, при обґрунтуванні стійкості потокових шифрів або їх елементів відносно статистичних атак потрібно знаходити лише нетривіальні нижні оцінки відносної відстані між заданою функцією та множиною лінійних функцій. На сьогодні є алгоритми [137, 138], які дозволяють статистично оцінювати зазначену відстань та базуються на ідеях роботи [134], але, на відміну від алгоритмів [17 – 22, 134 – 136], не формують самих лінійних наближень, що зменшує як трудомісткість оцінювання відносної відстані, так і обсяг потрібної пам’яті. Окремо відзначимо роботу [139], в якій описано алгоритм оцінювання максимальної незбалансованості білінійних наближень булевих функцій від n змінних за поліноміальний від n час.

В порівнянні з лінійними чи білінійними функціями, алгебраїчно вироджені функції утворюють суттєво більш широкий клас. Тому методи

аналізу властивостей наближень з цього класу булевих функцій є менш розвиненими.

Перші результати про кореляційні властивості алгебраїчно вироджених булевих функцій відносяться до 70-х років минулого століття [131]. В [29] досліджені наближення булевих функцій функціями з множини $B_{n,n-1}$; зокрема, отримано вираз для відносної відстані між довільною функцією $f: V_n \to \{0,1\}$ та множиною алгебраїчно вироджених функцій від n змінних, наведено спосіб знаходження функцій, найближчих до f у множині $B_{n,n-1}$ та отримано оцінки їх порядків (в [29, 132] порядком виродженості функції $g \in B_{n,n-1}$ називається найбільше число $n-k$, для якого g є k-вимірною функцією).

Вивченню k-вимірних наближень булевих функцій при всіх можливих значеннях k присвячено роботи [23, 24, 140 – 143], причому в [24] розглядаються функції над довільним скінченним полем. В [23] запропоновано ймовірнісний алгоритм розпізнавання k-вимірності. Для будь-якої функції $f: V_n \to \{0,1\}$, заданої за допомогою оракула, та чисел $k \in \overline{0, n-1}$, $\varepsilon \in (0,1)$ цей алгоритм дозволяє перевірити гіпотезу H_0: $f \in B_{n,k}$ проти альтернативи H_1: $d(f, B_{n,k}) \geq \varepsilon$ за $O(n 2^{2k} k\, \varepsilon^{-1})$ двійкових операцій. Більш ефективний тест k-вимірності, складність якого складає $O(n\, 2^k k^2\, \varepsilon^{-1})$ двійкових операцій, запропоновано в [141].

Для побудови ефективних атак на синхронні потокові шифри необхідно знаходити або обґрунтовувати відсутність k-вимірних функцій, достатньо близьких до заданої булевої функції f від n змінних. При цьому найбільший цікавим є випадок, коли функція f задається за допомогою оракула, число n є великим (наприклад, $n \geq 64$), а k – малим (фіксованим або повільно зростаючим з ростом n). Ефективність вирішення цієї задачі суттєво залежить від відстані до функції f та вимірності шуканих наближень.

Нехай g є k-вимірною функцією від n змінних, яка знаходиться від функції f на відносній відстані не більше $2^{-(k+1)}(1-\varepsilon)$, $\varepsilon \in (0,1)$ (зауважимо, що функція g визначається цією умовою однозначно). В [23] запропоновано ймовірнісний алгоритм, який дозволяє знаходити g за даними f, k і ε з ймовірністю не менше $1-\delta$, $\delta \in (0,1)$, за $O(2^{4k} n^2 \varepsilon^{-2} \log(2^{2k} n \delta^{-1}))$ двійкових операцій. В [142] представлено інший алгоритм, двійкова складність якого дорівнює $O(2^{2k} k^{-2} n^3 \varepsilon^{-2} \delta^{-1} \log(2^{2k} k^{-1} n \delta^{-1} \varepsilon^{-1}))$.

Суттєво більш складною є задача знаходження усіх функцій $g \in B_{n,k}$, розташованих від заданої функції $f : V_n \to \{0,1\}$ на відносній відстані не більше $2^{-k}(1-\varepsilon)$, $\varepsilon \in (0,1)$. Помітний внесок у розв'язання цієї задачі зроблено в роботі П. Гопалана [24], присвяченій дослідженню k-вимірних наближень функцій від n змінних над довільним скінченним полем. Одним з основних результатів цієї роботи є теорема, яка описує будову k-вимірних функцій g степеня не вище d, які знаходяться від заданої функції $f : V_n \to \{0,1\}$ на відносній відстані не більше $2^{-d}(1-\varepsilon)$, де $1 \le d \le k$, $\varepsilon \in (0,1)$. На основі цієї теореми в [24] запропоновано алгоритм побудови усіх зазначених функцій g зі складністю $O(2^n n M(k,d,\varepsilon))$, де $M(k,d,\varepsilon)$ є певною числовою функцією, що не залежить від n. На сьогодні алгоритм Гопалана [24] є одним з найефективніших універсальних алгоритмів побудови k-вимірних наближень булевих функцій, проте його складність швидко зростає з ростом k чи n. Відзначимо роботу [143], де отримано підсилення зазначеної вище теореми Гопалана, яке може бути використано для підвищення ефективності його алгоритму.

Інший метод побудови деяких алгебраїчно вироджених наближень булевих функцій, заданих за допомогою оракулів (так званий метод ймовірнісно нейтральних бітів), запропоновано в [10, 144]. Сутність методу полягає у

пошуку наближень функції $f : V_n \to \{0,1\}$ серед її підфункцій, які отримуються шляхом фіксації константами змінних x_i, для яких число одиниць у векторі значень похідної

$$D_i f = f(x_1,..., x_{i-1}, x_i, x_{i+1}, ..., x_n) \oplus f(x_1,... , x_{i-1}, x_i \oplus 1, x_{i+1}, ..., x_n)$$

не перевищує заданої (невеликої) межі. На конкретних прикладах показано, що такі наближення приводять до більш ефективних в порівнянні з повним перебором ключів атак на редуковані версії СПШ Grain-128 і Trivium, але не досліджена ефективність методу побудови наближень в загальному випадку. Таке дослідження проведено в [145], де отримано досяжні верхні межі відносної відстані між булевою функцією f та найближчою до неї функцією, що не залежить від змінних з номерами із заданої множини, а також між функцією f та її наближеннями, що отримуються методом ймовірнісно нейтральних бітів. Крім того, показано, що у певних випадках, зазначений метод приводить до наближень, які є далекими від найкращих. Як наслідок, задачі ефективного знаходження (чи обґрунтування відсутності) алгебраїчно вироджених наближень булевих функцій від декількох десятків або сотень змінних залишаються у загальному випадку не вирішеними.

В цілому, проведений аналіз дозволяє зробити такі висновки:

1) для побудови переважної більшості сучасних атак на СПШ на основі алгебраїчно вироджених наближень булевих функцій використовуються дуже вузькі класи наближень (афінні функції або функції від малого числа змінних);

2) за винятком окремих спеціальних випадків [10, 17 – 24], відсутні ефективні алгоритми пошуку наближень булевих функцій із зазначених класів;

3) відсутні методи, які б дозволяли за достатньо загальних умов обґрунтовувати стійкість СПШ відносно зазначених вище атак.

Наведені факти свідчать про певне протиріччя між потребами в обґрунтовано стійких та практичних алгоритмах потокового шифрування, що необхідні для забезпечення безпеки державних інформаційних ресурсів, з одного боку, та відсутністю методів обґрунтування стійкості цих алгоритмів відносно широкого кола сучасних атак, з іншого. Зазначене протиріччя визначає напрямок, характер та окремі задачі дисертаційного дослідження.

1.4. Основні напрями та окремі задачі дисертаційного дослідження

Вище показана актуальність *наукової задачі* дисертаційної роботи, яка полягає в розробці методу побудови науково обґрунтованих оцінок параметрів, що характеризують стійкість синхронних потокових шифрів відносно статистичних атак на основі алгебраїчно вироджених наближень булевих функцій.

Метою дисертаційної роботи є підвищення ефективності використання національних інформаційних ресурсів за рахунок зменшення часу проведення експертних досліджень алгоритмів потокового шифрування, призначених для захисту інформації в спеціальних інформаційно-телекомунікаційних системах України.

Об`єктом дослідження в роботі є криптографічні перетворення інформації, які реалізуються синхронними потоковими шифрами, призначеними для захисту інформації в спеціальних інформаційно-телекомунікаційних системах України, а *предметом дослідження* – методи оцінювання та обґрунтування стійкості синхронних потокових шифрів відносно статистичних атак на основі алгебраїчно вироджених наближень булевих функцій.

У відповідності до поставленої мети, наукова задача дисертаційної роботи включає в себе ряд взаємопов’язаних окремих задач, порядок розв’язання яких визначає основні напрями дисертаційних досліджень (рис. 1.8).

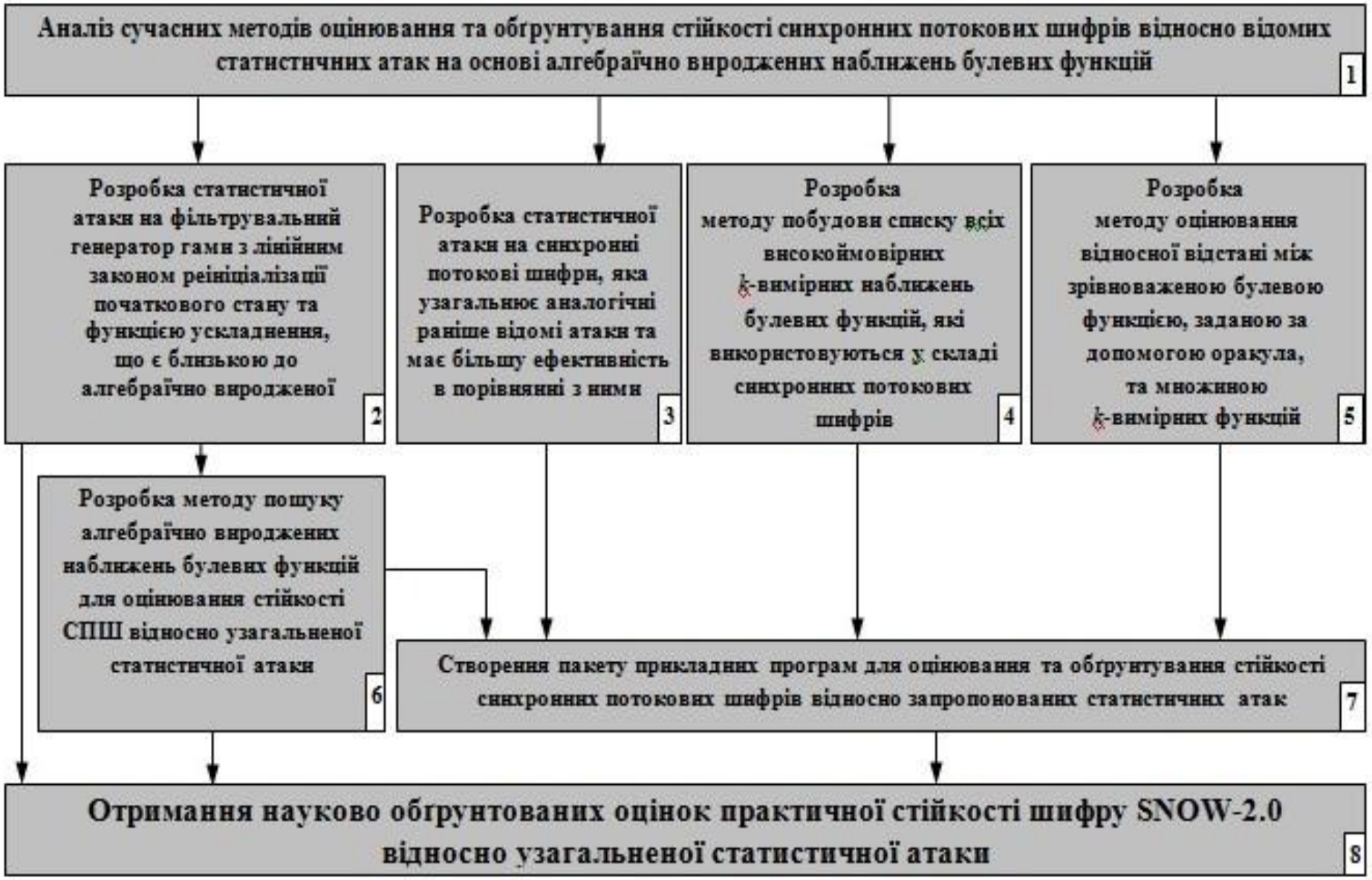

Рис. 1.8. Окремі задачі досліджень

Перший напрям полягає в побудові статистичних атак на СПШ, які узагальнюють раніше відомі атаки, а також оцінюванні практичної стійкості шифру SNOW-2.0, який є прототипом національного стандарту потокового шифрування України (задачі 2, 3, 7, 8 на рис. 1.8). Основною задачею *другого напряму* є розробка методів пошуку чи обґрунтування відсутності алгебраїчно вироджених наближень булевих функцій, що використовуються при побудові зазначених статистичних атак (задачі 4, 5, 6 на рис. 1.8).

Вирішення перелічених окремих задач дозволяє вирішити загальну наукову задачу дисертаційної роботи та досягнути поставленої в роботі мети.

Висновки

1. Важливою задачею забезпечення інформаційної безпеки держави є створення та підтримання умов, що гарантують безпечну обробку, зберігання та передачу державних інформаційних ресурсів. Вирішення цієї задачі неможливо без застосування криптографічних методів та засобів, серед яких достатньо широке застосування в спеціальних інформаційно-телекомунікаційних системах знаходять потокові шифри. Це обумовлено особливостями побудови та функціонування потокових шифрів, основною перевагою яких є надвисока швидкість шифрування. Переважну більшість систем потокового шифрування складають синхронні потокові шифри.

2. Основним принципом побудови сучасних потокових шифрів є умова їх практичної стійкості відносно всіх відомих атак в поєднанні з можливістю ефективної реалізації алгоритмів потокового шифрування на різноманітних платформах. Найбільш потужними атаками на потокові шифри є статистичні атаки на основі підібраних векторів ініціалізації [9 – 16, 25 – 28]. Стійкість СПШ відносно зазначених атак характеризується відсутністю певних алгебраїчно вироджених наближень булевих функцій, пов’язаних з

алгоритмами шифрування. Задача пошуку чи обґрунтування відсутності таких наближень є обчислювально складною і для її розв’язання (за виключенням окремих спеціальних випадків [10, 17 – 24]) не відомо ефективних алгоритмів.

3. Для побудови переважної більшості сучасних атак на СПШ на основі алгебраїчно вироджених наближень булевих функцій використовуються дуже вузькі класи наближень. Крім того, відсутні методи, які б дозволяли за достатньо загальних умов обґрунтовувати стійкість СПШ відносно зазначених вище атак. Залишається невирішеною задача обґрунтування практичної стійкості шифру SNOW-2.0, що є прототипом майбутнього національного стандарту потокового шифрування, відносно атаки, описаної в [10].

4. У відповідності до поставленої мети, розв’язання наукової задачі дисертаційної роботи включає в себе дослідження за двома напрямами. Перший напрям полягає в побудові статистичних атак, які узагальнюють раніше відомі атаки на основі алгебраїчно вироджених наближень булевих функцій [9, 10, 25 – 28], а також обґрунтування стійкості шифру SNOW-2.0 відносно таких атак. Основною задачею другого напряму є розробка методів пошуку чи обґрунтування відсутності алгебраїчно вироджених наближень булевих функцій, що використовуються при побудові зазначених атак.

РОЗДІЛ 2

СТАТИСТИЧНІ АТАКИ НА СИНХРОННІ ПОТОКОВІ ШИФРИ НА ОСНОВІ АЛГЕБРАЇЧНО ВИРОДЖЕНИХ НАБЛИЖЕНЬ БУЛЕВИХ ФУНКЦІЙ

В попередньому розділі показано, що для розв’язання наукової задачі дисертаційної роботи необхідно, перш за все, уніфікувати та узагальнити відомі атаки на синхронні потокові шифри на основі підібраних векторів ініціалізації [9 – 16, 25 – 28]. Зокрема, необхідно точно сформулювати загальні умови застосовності цих атак і отримати аналітичні оцінки параметрів, що характеризують їх ефективність (трудомісткості та обсягу матеріалу, необхідного для успішної реалізації атаки з заданою надійністю). Даний розділ присвячено розв’язанню зазначених задач.

У підрозділі 2.1 описано статистичну атаку на генератори гами з лінійним законом реініціалізації початкового стану та функцією ускладнення, що є близькою до алгебраїчно виродженої функції. На відміну від раніше відомих [9, 10, 25 – 28], ця атака є застосовною до більш широкого класу генераторів гами за менш жорстких обмежень відносно їх функцій ускладнення. Отримано аналітичну верхню межу трудомісткості атаки та показано, що за певних умов її може бути успішно застосовано на практиці до генераторів гами з довжиною ключа, принаймні, 128 біт.

У підрозділі 2.2 викладено більш загальну статистичну атаку на синхронні потокові шифри, яка узагальнює атаку FKM [10], а також кубічну атаку [9] і базується на наближеннях булевих функцій, що реалізуються алгоритмами шифрування, алгебраїчно виродженими функціями. Охарактеризовано клас ключів, які можна відновити за допомогою описаної атаки, та наведено явний вираз її трудомісткості як функції її надійності, складності обчислення значення функції-оракула та алгоритму опробування ключів (відзначимо, що аналоги зазначених результатів є невідомими навіть для атаки FKM).

Крім того, запропоновано ймовірнісний алгоритм побудови булевих функцій від меншої кількості змінних, що використовуються для визначення наближень при проведенні узагальненої статистичної атаки. Цей алгоритм є модифікацією (детермінованого) алгоритму з [10], проте, на відміну від останнього, має теоретичне обґрунтування.

Нарешті, на прикладі редукованої версії шифру Grain-128 показано, що узагальнена статистична атака може бути майже в 2^{27} разів ефективніше у порівнянні з аналогічною раніше відомою атакою [10].

2.1. Статистична атака на генератор гами з лінійним законом реініціалізації початкового стану та функцією ускладнення, що є близькою до алгебраїчно виродженої

2.1.1. Алгоритм відновлення ключа генератора гами та його теоретичне обґрунтування. Нижче використовуються такі позначення: V_n – множина двійкових векторів довжини n, $F_{m\times n}$ – множина матриць розміру $m\times n$ над полем $F=\mathbf{GF}(2)$, $\overline{a,b}=\{i\in\mathbf{Z}: a\le i\le b\}$, де $a,b\in\mathbf{Z}$.

Розглянемо генератор гами (рис. 2.1), функціонування якого в режимі реініціалізації початкового стану описується такою системою рівнянь:

$$x^{(j)}=Ak\oplus Bc^{(j)},\ f(L^i(x^{(j)}))=\gamma_i^{(j)},\ i\in\overline{0,T-1},\ j\in\overline{1,r}. \qquad (2.1)$$

Тут $x^{(j)}\in V_N$ – початковий стан генератора при j-му запуску, який відповідає ключу $k\in V_{l_0}$ та вектору ініціалізації $c^{(j)}\in V_{l_1}$; A та B – булеві матриці; $f=f(x_1,\dots,x_N)$ – функція ускладнення генератора, яка суттєво залежить від n змінних $x_{\mu_1},\dots,x_{\mu_n}$, де $1\le\mu_1<\dots<\mu_n\le N$; $L:V_N\to V_N$ – невироджене

лінійне перетворення над полем F; $\gamma_i^{(j)}$ – знак шифрувальної гами в i-му такті при j-му запуску генератора гами, $i \in \overline{0, T-1}$, $j \in \overline{1, r}$.

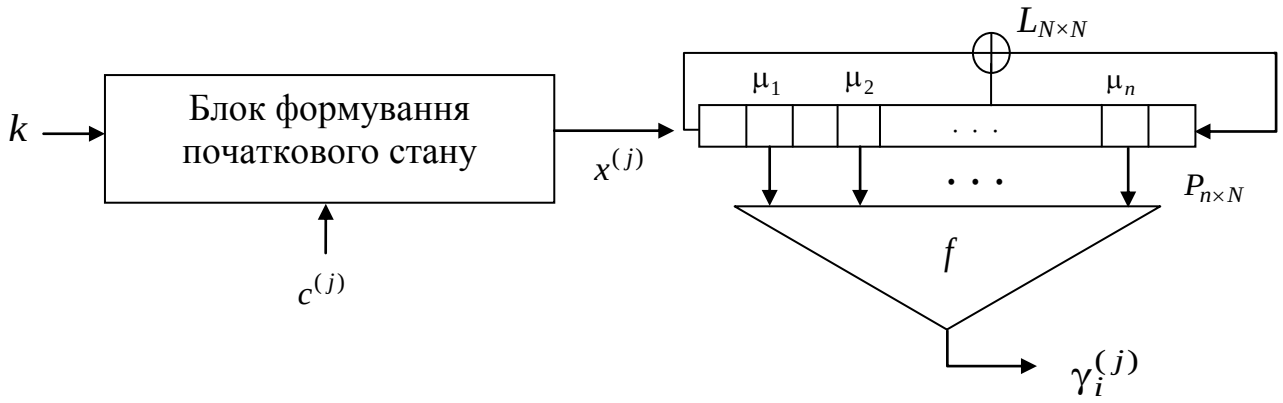

Рис. 2.1. Схематичне зображення генератора гами з лінійним законом реініціалізації початкового стану

Необхідно відновити ключ k за відомими значеннями f, L, A, B, $\gamma_i^{(j)}$, $c^{(j)}$, $i \in \overline{0, T-1}$, $j \in \overline{1, r}$.

Позначимо символом P $n \times N$-матрицю над полем F таку, що $Px^{\mathrm{T}} = (x_{\mu_1}, \ldots, x_{\mu_n})^{\mathrm{T}}$ для будь-якого $x = (x_1, \ldots, x_N) \in V_N$, покладемо $A_i = PL^i A$, $B_i = PL^i B$, $i = 0, 1, \ldots$, та запишемо систему рівнянь (2.1) у вигляді

$$f(A_i k \oplus B_i c^{(j)}) = \gamma_i^{(j)},\ i \in \overline{0, T-1},\ j \in \overline{1, r}. \qquad (2.2)$$

Припустимо, що виконані такі умови.

1. Існує функція $g(x) = \varphi(Mx)$, $x \in V_n$, де $\varphi : V_s \to \{0, 1\}$, $M \in F_{s\times n}$, така, що

$$\mathbf{P}\{f(X) = g(X)\} = p \geq 1/2 \cdot (1 + \varepsilon),\ \varepsilon \in (0, 1), \qquad (2.3)$$

де X – випадковий рівноймовірний двійковий вектор довжини n; іншими словами, для функції f існує s-вимірний статистичний аналог g, який знаходиться від f на відстані не більше $2^{n-1}(1-\varepsilon)$, де $\varepsilon \in (0,1)$;

2. Існує множина $I = \{i_1, i_2, \ldots, i_t\} \subseteq \overline{0, T-1}$ така, що

$$\text{rank}(B_{i_1}) = \text{rank}(B_{i_2}) = \ldots = \text{rank}(B_{i_t}) = n \le l_1 \qquad (2.4)$$

та

$$\text{rank}\begin{pmatrix} MA_{i_1} \\ MA_{i_2} \\ \ldots\ldots \\ MA_{i_t} \end{pmatrix} = l_0 \qquad (2.5)$$

(нагадаємо, що l_0 та l_1 позначають відповідно довжину ключа та вектора ініціалізації генератора гами).

За цих умов можна запропонувати алгоритм відновлення ключа генератора гами (алгоритм 2.1), зображений на рис. 2.2.

Зауважимо, що на підставі рівності (2.5) система рівнянь, яка будується на третьому етапі алгоритму 2.1, має не більше одного розв'язку. Тому якщо зазначена система є сумісною, то ключ буде відновлено однозначно.

Для оцінки ефективності алгоритму 2.1 зробимо два додаткових припущення відносно функціонування генератора гами.

Перш за все, припустимо, що вектори ініціалізації $c^{(1)}, c^{(2)}, \ldots, c^{(r)}$ є незалежними в сукупності, випадковими та рівномірно розподіленими на множині V_{l_1}.

Алгоритм 2.1

Вхідні дані: f, L, A, B, $\gamma_i^{(j)}$, $c^{(j)}$ ($i \in \overline{0, T-1}$, $j \in \overline{1, r}$), φ, M.

Етап 1 (*попередні обчислення*). Побудувати множину I, яка задовольняє умові 2; обчислити матриці MA_i, MB_i, $i \in I$.

Етап 2 (*застосування методу максимальної правдоподібності*). Для кожного $i \in I$ обчислити значення

$$\nu_i(y) = \sum_{j=1}^{r} (\varphi(y \oplus MB_i c^{(j)}) \oplus \gamma_i^{(j)}), \ y \in V_s \qquad (2.6)$$

та знайти вектор $\hat{y}_i \in V_s$, такий, що $\nu_i(\hat{y}_i) = \min\limits_{y \in V_s} \nu_i(y)$.

Етап 3 (*відновлення ключа*). Скласти систему лінійних рівнянь

$$MA_i k = \hat{y}_i, \ i \in I \qquad (2.7)$$

відносно ключа k та, розв'язуючи її, знайти цей ключ.

Рис. 2.2. Алгоритм відновлення ключа генератора гами з лінійним законом реініціалізації початкового стану

Для будь-яких $i \in I$, $y \in V_s$ розглянемо події $\Omega_i^{(j)}(y) = \{\varphi(y \oplus MB_i c^{(j)}) \oplus \gamma_i^{(j)} = 1\}$, $j \in \overline{1, r}$. Зауважимо, що ці події є незалежними в сукупності для кожної пари значень $i \in I$, $y \in V_s$. Крім того, на підставі умови (2.4) випадкові вектори $B_i c^{(j)}$, $j \in \overline{1, r}$, рівномірно розподілені на множині V_n. Звідси в силу формул (2.2), (2.3) випливає, що при $y = MA_i k$ справедливі такі співвідношення:

$$\mathbf{P}\left(\Omega_i^{(j)}(MA_ik)\right)=\mathbf{P}\{\varphi(MA_ik\oplus MB_ic^{(j)})\oplus\gamma_i^{(j)}=1\}=$$

$$=\mathbf{P}\{g(A_ik\oplus B_ic^{(j)})\oplus f(A_ik\oplus B_ic^{(j)})=1\}=$$

$$=\mathbf{P}\{g(X)\oplus f(X)=1\}=1-p\le 1/2\cdot(1-\varepsilon).$$

Отже, для будь-яких $i\in I$, $j\in\overline{1,r}$ справедливе таке співвідношення:

$$\mathbf{P}\left(\Omega_i^{(j)}(MA_ik)\right)\le 1/2\cdot(1-\varepsilon). \qquad (2.8)$$

Приймемо тепер, як друге припущення відносно функціонування генератора гами, що

$$\mathbf{P}\left(\Omega_i^{(j)}(y)\right)=1/2 \qquad (2.9)$$

для будь-яких $i\in I$, $j\in\overline{1,r}$ та $y\ne MA_ik$.

Позначимо P_e ймовірність помилки алгоритму 2.1, тобто ймовірність події, яка полягає в тому, що алгоритм не відновить шуканий ключ k.

Лема 2.1. За припущень, зазначених вище, справедлива нерівність

$$P_e\le 2^s t\exp\{-1/8\cdot r\,\varepsilon^2\}. \qquad (2.10)$$

Доведення. Якщо алгоритм припускається помилки, то порушується хоча б одна з рівностей (2.7). Отже, $P_e\le\sum_{i\in I}\mathbf{P}\{MA_ik\ne\hat{y}_i\}\le t\max_{i\in I}\mathbf{P}\{MA_ik\ne\hat{y}_i\}$ і для доведення формули (2.10) достатньо переконатися в справедливості таких нерівностей:

$$\mathbf{P}\{MA_i k \neq \hat{y}_i\} \leq 2^s \exp\{-1/8 \cdot r\, \varepsilon^2\},\ i \in I. \qquad (2.11)$$

Зафіксуємо $i \in I$. Зауважимо, що в силу означення вектора $\hat{y}_i$ для будь-якого $C > 0$ справедливі співвідношення

$$\{MA_i k \neq \hat{y}_i\} \subseteq \{\nu_i(MA_i k) \geq C\} \cup \bigcup_{y \in V_s : y \neq MA_i k} \{\nu_i(y) < C\},$$

з яких випливає, що

$$\mathbf{P}\{MA_i k \neq \hat{y}_i\} \leq \mathbf{P}\{\nu_i(MA_i k) \geq C\} + (2^s - 1) \max_{y \in V_s : y \neq MA_i k} \mathbf{P}\{\nu_i(y) < C\}. \qquad (2.12)$$

Далі, згідно з рівністю (2.6), $\nu_i(MA_i k)$ є сумою незалежних випадкових величин $\xi_j = \varphi(MA_i k \oplus MB_i c^{(j)}) \oplus \gamma_i^{(j)}$, $j \in \overline{1, r}$. Отже, вважаючи

$$C = 1/4 \cdot r(2 - \varepsilon), \qquad (2.13)$$

на підставі формули (2.8) та нерівності для ймовірності великих ухилень (див. наприклад, [156], с. 31) отримаємо такі співвідношення:

$$\mathbf{P}\{\nu_i(MA_i k) \geq C\} = \mathbf{P}\{\sum_{i=1}^{r} \xi_i - \sum_{i=1}^{r} \mathbf{E}\xi_i \geq C - 1/2 \cdot r(1-\varepsilon)\} \leq \mathbf{P}\{\sum_{i=1}^{r} \xi_i - \sum_{i=1}^{r} \mathbf{E}\xi_i \geq 1/4 \cdot r\varepsilon\} \leq$$

$$\leq \exp\{-1/8 \cdot r\varepsilon^2\}. \qquad (2.14)$$

Нехай зараз $y \in V_s$, $y \neq MA_i k$; тоді в силу нерівностей (2.6), (2.9) $\nu_i(y)$ є сумою незалежних випадкових величин $\eta_j = \varphi(y \oplus MB_i c^{(j)}) \oplus \gamma_i^{(j)}$, $j \in \overline{1,r}$, кожна з яких розподілена рівномірно на множині $\{0,1\}$. Отже на підставі формули (2.13) та нерівності для ймовірності великих ухилень

$$\mathbf{P}\{\nu_i(y) < C\} = \mathbf{P}\{\sum_{i=1}^{r}\eta_i - \sum_{i=1}^{r}\mathbf{E}\eta_i < C - 1/2 \cdot r\} = \mathbf{P}\{\sum_{i=1}^{r}\eta_i - \sum_{i=1}^{r}\mathbf{E}\eta_i < -1/4 \cdot r\varepsilon\} \leq$$

$$\leq \exp\{-1/8 \cdot r\varepsilon^2\}. \qquad (2.15)$$

Підставляючи оцінки (2.14), (2.15) у формулу (2.12), отримуємо нерівність (2.11). Лему доведено.

Приймемо в якості елементарної довільну двійкову операцію (булеву функцію двох змінних), а також операцію вигляду $i \mapsto i+1$, де i – будь-яке невід'ємне ціле число.

Наступна теорема дозволяє оцінити ефективність алгоритму 2.1.

Теорема 2.1. Нехай виконуються зазначені вище припущення щодо генератора гами, $\delta \in (0,1)$ та

$$r = \lceil 8 \cdot \varepsilon^{-2} \ln (2^s t\, \delta^{-1}) \rceil. \qquad (2.16)$$

Тоді алгоритм 2.1 відновлює ключ k з ймовірністю не менше ніж $1-\delta$, використовуючи (без урахування передобчислень на першому етапі)

$$\tau(s,\varepsilon) = O\left((2^s l_1 r + l_0^2)ts\right) \qquad (2.17)$$

елементарних операцій. При цьому для виконання алгоритму потрібно

$$\lambda(s,\varepsilon) = ri_t \tag{2.18}$$

знаків гами, де i_t – найбільший елемент множини I.

Доведення. Перше твердження теореми випливає безпосередньо зі співвідношень (2.10), (2.16), а останнє – з опису алгоритму. Нарешті, оскільки для знаходження векторів $\hat{y}_i$ $(i \in I)$ потрібно $O(2^s tsl_1 r)$ елементарних операцій, а для розв'язку системи лінійних рівнянь (2.7) методом Гауса – $O(l_0^2 ts)$ операцій, то трудомісткість алгоритму складає $O\left((2^s l_1 r + l_0^2)ts\right)$ елементарних операцій.

Наслідок 2.1. Нехай за умови теореми $l_0 = O(s)$, $l_1 = O(s)$ при $s \to \infty$,

$$t = \left\lceil (l_0 + C)s^{-1} \right\rceil. \tag{2.19}$$

де $C \geq 0$. Тоді асимптотична складність алгоритму при $s \to \infty$ та $\varepsilon \to 0$ складає $O\left(2^s s^4 \varepsilon^{-2}\right)$ елементарних операцій.

Опишемо докладніше процедуру побудови множини I на першому етапі алгоритму 2.1. Нагадаємо, що ця множина складається з чисел $i_1, i_2, \ldots, i_t$, для яких виконуються співвідношення (2.4) та (2.5), причому, згідно з останнім з них, t повинно бути не менше ніж $\left\lceil l_0 s^{-1} \right\rceil$.

Назвемо число $i = 0, 1, \ldots$ *допустимим*, якщо $\operatorname{rank}(B_i) = n$, та *недопустимим* – в протилежному випадку.

З рівності $B_i = PL^i B$, $i = 0, 1, \ldots$, випливає, що за умови $N \leq l_1$, $\operatorname{rank}(B) = N$ кожне значення $i = 0, 1, \ldots$ є допустимим і можна покласти $I = \{0, 1, \ldots, t-1\}$, вибираючи в якості t найменше натуральне число, для якого

виконується рівність (2.15). Якщо ж $n \le l_1 < N$, то не всі значення i є допустимими, і процедура побудови множини I, в принципі, може виявитися занадто складною. Разом з тим, як показує наступне твердження, в багатьох практично важливих випадках "питома вага" недопустимих значень є вельми невеликою.

Твердження 2.1. Нехай L є повноцикловим лінійним перетворенням над полем F, $n < l_1 < N$ та $\mathrm{rank}(B) = l_1$. Тоді ймовірність того, що обране випадково та рівноймовірно з множини $\overline{0, 2^N - 1}$ число i є недопустимим, є менше ніж $2^{-(l_1 - n)}$.

Доведення. Позначимо $J = \{i \in \overline{0,2^N - 1} : \mathrm{rank}(PL^i B) < n\}$ сукупність всіх недопустимих чисел з множини $\overline{0, 2^N - 1}$; U – підпростір векторного простору V_n, породжений рядками матриці P; W – підпростір, дуальний до векторного простору, породженому стовпцями матриці B. З наданих означень випливає, що $i \in J \Leftrightarrow (\exists\, x \in V_n \setminus \{0\} : xPL^i B = 0) \Leftrightarrow L^i(U) \cap W \neq \{0\}$.

Розглянемо таблицю, яка складається з $2^n - 1$ рядків, занумерованих векторами $u \in U \setminus \{0\}$, та $2^{N - l_1} - 1$ стовпців, занумерованих векторами $w \in W \setminus \{0\}$, що містить на перетині довільного рядка u та довільного стовпця w єдине число $i(u, w) \in \overline{0, 2^N - 1}$ таке, що $uL^{i(u,w)} = w$. Зрозуміло, що ця таблиця є латинським прямокутником, а $|J|$ є числом різних елементів у ній. Отже, $|J| \le (2^n - 1)(2^{N - l_1} - 1) < 2^{N - l_1 + n}$ та, значить, $2^N |J| < 2^{-(l_1 - n)}$.

Твердження доведено.

2.1.2. Експериментальне дослідження статистичної атаки на генератор гами з лінійним законом реініціалізації початкового стану. Для отримання чисельних значень параметрів, що характеризують ефективність запропонованої атаки, та оцінки можливості її

застосування на практиці проведено низку обчислювальних експериментів. В даному пункті наведено типові результати, отримані в ході таких експериментів для таких значень вхідних параметрів: довжина початкового стану генератора $N = 128$; довжина ключа $l_0 = 128$; число невідомих функції ускладнення $n = 64$. Для визначення лінійної функції переходів (матриці L) використовувався примітивний над полем F поліном $x^{128} \oplus x^{95} \oplus x^{57} \oplus x^{45} \oplus x^{38} \oplus x^{36} + 1$, а для визначення суттєвих змінних функції ускладнення (матриці P) – фіксований набір констант $1 \le \mu_1 < ... < \mu_n \le N$.

Дані на рис. 2.3 отримані в результаті 200 незалежних випробувань, в j-му з яких генерувалась випадкова рівноймовірна 128×64-матриця B та

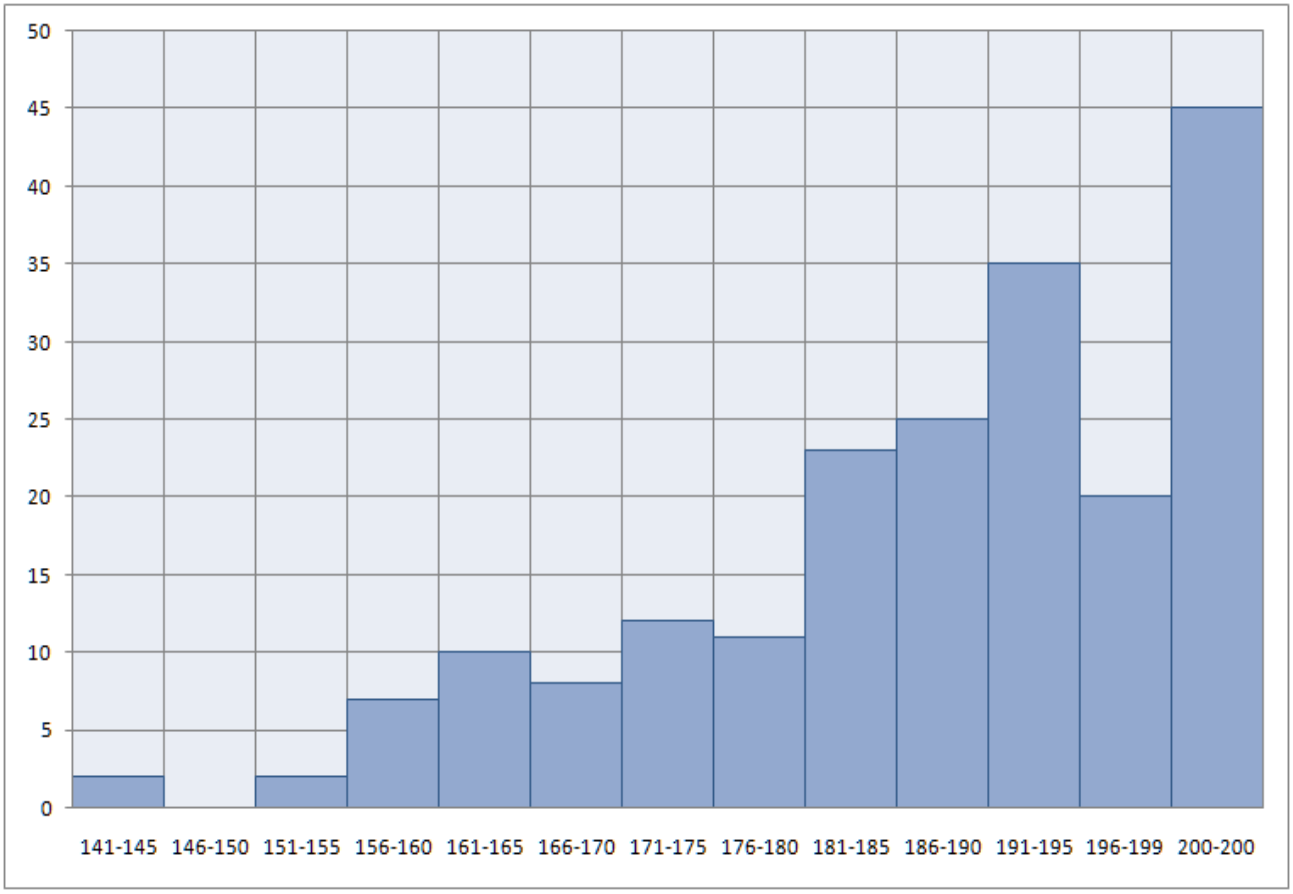

Рис. 2.3. Емпірічний розподіл числа допустимих значень значень випадкової величини ν_j в інтервалі [0, 199]

підраховувалось значення випадкової величини $\nu_j = \#\{i \in \overline{0,199} : \mathrm{rank}(B_i) = n\}$. Для кожного інтервалу $[u, v]$ на рисунку показано кількість тих значень $j \in \overline{1,200}$, для яких $u \le \nu_j \le v$.

Отримані результати свідчать про те, що для випадково згенерованої матриці B допустимі числа з'являються достатньо часто. Наприклад, в 35 з 200 проведених випробуваннях кількість допустимих чисел $i \in \overline{0,199}$ складає від 191 до 195. Ефект "повного рангу" (коли всі значення є допустимими або, що те ж саме всі матриці B_i мають ранг n) спостерігається приблизно в 25% випадків (45 з 200).

В табл. 2.1 наведено результати експериментальної перевірки умови 2 (див. с. 45) та оцінювання обсягу матеріалу (кількості знаків гами при фіксованих значеннях ключа та вектора ініціалізації), необхідного для проведення атаки.

Таблиця 2.1

Результати експериментальної оцінки необхідного обсягу матеріалу при фіксованих значеннях ключа та вектора ініціалізації

s	l_1	Параметр	Обсяг матеріалу		
2	64	t^*	64	64	64
		$i_{\min}$	64	64	64
		$i_{\max}$	117	100	118
		i_{av}	76	74	76
10	80	t^*	13	13	13
		$i_{\min}$	13	13	13
		$i_{\max}$	33	32	38
		i_{av}	18	17	18
20	120	t^*	8	8	8
		$i_{\min}$	11	11	11
		$i_{\max}$	11	11	11
		i_{av}	11	11	11

Дані отримано в результаті 100 незалежних випробувань, в кожному з яких генерувався випадковий набір, що складається з незалежних рівноймовірних матриць A, B та M відповідного розміру, та формувалася множина I найменшої потужності, що задовольняє умові (2.2). В таблиці зазначено такі параметри: t^* – значення числа t з найбільшою частотою зустрічальності (серед 100 проведених випробувань); $i_{\min}$, $i_{\max}$ та i_{av} – відповідно найменше, найбільше та середнє значення параметра i_t (найбільшого елемента множини I) при $t = t^*$. Зауважимо, що саме від останнього параметра залежить обсяг матеріалу, необхідного для проведення атаки (див. формулу (2.18)).

Як видно з табл. 2.1, при випадковому незалежному та рівноймовірному виборі матриць A, B, M середня кількість знаків гами при кожному запуску генератора, що є необхідною для проведення атаки, практично не залежить від довжини вектора ініціалізації та коливається від 11 до 76 в залежності від числа s невідомих функції φ (див. умову 1 на с. 44). При цьому потужність множини I дорівнює приблизно $\lceil l_0 s^{-1} \rceil$.

В табл. 2.2, 2.3 наведено чисельні значення параметрів (2.16) та (2.17) відповідно, обчислені з використанням даних з табл. 2.1 при $\delta = 0{,}01$ та $l_0 = 128$.

Як видно з таблиць, при $\varepsilon \geq 0{,}05$ та $s \leq 10$ атака, в принципі, дозволяє відновлювати ключ довжиною $l_0 = 128$ біт зі складністю не більше 2^{38} елементарних операцій, використовуючи не більше 2^{15} відрізків гами (довжиною не більше 117 знаків кожен) разом з відповідними їм векторами ініціалізації.

Алгоритм 2.1 було реалізовано програмно та застосовано до низки генераторів гами з певними функціями ускладнення, що є близькими до s-вимірних функцій (табл. 2.4).

Таблиця 2.2

Кількість векторів ініціалізації, достатня для відновлення ключа з надійністю не менше ніж 99 %

Параметри			r
ε	s	t^*	
0,05	2	64	2^{14}
	10	13	2^{15}
	20	8	2^{16}
0,10	2	64	2^{12}
	10	13	2^{13}
	20	8	2^{14}
0,30	2	64	2^{9}
	10	13	2^{10}
	20	8	2^{10}

Таблиця 2.3

Чисельні оцінки трудомісткості запропонованої атаки

Параметри				$\tau(s,\varepsilon)$
l_1	s	t^*	r	
64	2	64	2^{14}	2^{29}
	10	13	2^{15}	2^{38}
	20	8	2^{16}	2^{49}
80	2	64	2^{12}	2^{28}
	10	13	2^{13}	2^{36}
	20	8	2^{14}	2^{47}
120	2	64	2^{9}	2^{25}
	10	13	2^{10}	2^{34}
	20	8	2^{10}	2^{45}

Під час моделювання використовувались генератори гами з параметрами N, l_0, n, а також матриці L та P, зазначені вище. Матриці A та B, що визначають закон реініціалізації генератора гами, були згенеровані заздалегідь та не змінювалися в ході обчислювальних експериментів.

Таблиця 2.4

Результати застосування алгоритму 2.1 для відновлення ключа генератора гами

Параметри					T_{av}, с
s	ε	i_t	r	r_{pr}	
2	0,05	63	25113	11000	1
	0,10		6279	2650	1
	0,30		698	180	1
10	0,05	12	37757	75600	4439
	0,10		9440	18900	1132
	0,30		1049	2350	155

Функція ускладнення f генератора задавалася окремо для кожної пари чисел (s, ε), зазначених в табл. 2.4, за таким правилом: $f(x) = \varphi_s(M_s x) \oplus \xi_x$, $x \in V_n$, де

$$\varphi_2(x_1, x_2) = x_1 x_2,\ (x_1, x_2) \in V_2;$$

$$\varphi_{10}(x_1, \ldots, x_{10}) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_9 \oplus$$

$$\oplus x_2x_{10} \oplus x_4x_8 \oplus x_5x_6 \oplus x_6x_8 \oplus x_6x_{10} \oplus x_7x_8 \oplus x_8x_9 \oplus x_8x_{10} \oplus$$

$$\oplus x_9x_{10} \oplus x_1x_2x_8 \oplus x_1x_4x_{10} \oplus x_1x_8x_9 \oplus x_1x_9x_{10} \oplus x_2x_3x_8 \oplus x_2x_4x_8 \oplus x_2x_4x_{10} \oplus x_2x_7x_8 \oplus$$

$$\oplus x_2x_8x_{10} \oplus x_2x_9x_{10} \oplus x_3x_4x_8 \oplus x_3x_8x_9 \oplus x_4x_7x_8 \oplus x_4x_8x_9 \oplus x_5x_6x_8 \oplus x_5x_6x_{10} \oplus x_6x_8x_{10} \oplus x_7x_8x_9 \oplus$$

$$\oplus x_8x_9x_{10} \oplus x_1x_2x_3x_8 \oplus x_1x_2x_7x_8 \oplus x_1x_3x_5x_8 \oplus x_1x_3x_8x_9 \oplus x_1x_4x_8x_{10} \oplus$$

$$\oplus x_1x_5x_7x_8 \oplus x_1x_7x_8x_9 \oplus x_1x_8x_9x_{10} \oplus x_2x_3x_4x_8 \oplus x_2x_3x_5x_8 \oplus x_2x_4x_7x_8 \oplus x_2x_4x_8x_{10} \oplus x_2x_5x_7x_8 \oplus$$

$$\oplus\, x_2x_8x_9x_{10} \oplus x_3x_4x_8x_9 \oplus x_4x_7x_8x_9 \oplus x_5x_6x_8x_{10},\ (x_1, \ldots, x_{10}) \in V_{10};$$

M_s є певною $s \times n$-матрицею над полем F, а $(\xi_x : x \in V_n)$ є випадковим двійковим вектором з незалежними координатами, що задовольняють умові $\mathbf{P}\{\xi_x = 0\} \geq 1/2 \cdot (1+\varepsilon)$, $x \in V_n$.

На етапі попередніх обчислень за відомими матрицями A, B, M_s будувалася множина I потужності t, яка задовольняє умові 2 (див. табл. 2.4, де для кожного $s \in \{2, 10\}$ наведено значення найбільшого елемента i_t цієї множини; при цьому $t = i_t + 1$).

Далі при $s = 2$ для кожного ε, наведеного в табл. 2.4, вибиралося натуральне число r' і 30 разів виконувалася така процедура:

– генерувався випадковий ключ $k \in V_{128}$ та послідовність (випадкових, незалежних один від одного та від ключа) векторів ініціалізації $c^{(j)} \in V_{l_1}$, за якими обчислювалися знаки гами $\gamma_i^{(j)}$ ($i \in I$, $j \in \overline{1, r'}$);

– виконувався етап 2 алгоритму 2.1 і формувалася система лінійних рівнянь вигляду (2.7).

Вважалося, що ця процедура завершується успішно, якщо ключ k задовольняє усім рівнянням отриманої системи. Обчислення проводилися на ПК з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам’яті 4 ГБ RAM на базі 32-розрядної ОС Windows 7 Service Pack 1.

Середній час роботи програми займає від декількох секунд до декількох десятків хвилин в залежності від значень параметрів s та ε.

Значення r в табл. 2.4 обчислені за формулою (2.16) при $\delta = 0{,}1$. В передостанній колонці табл. 2.4 наведено (знайдені експериментальним шляхом) оцінки найменшого значення r_{pr} параметра r', за якого усі 30 запусків процедури є успішними, а в останній колонці – середній (за всіма 30

запусками) час виконання етапу 2 алгоритму 2.1. Дані при $s = 10$ отримано аналогічно, лише з тією відмінністю, що замість 30 запусків процедури (для кожного ε з табл. 2.4) виконувалося тільки 15. Остання обставина обумовлена помітним зростанням часу виконання алгоритму 2.1 з ростом параметра s.

Як видно з табл. 2.4, для успішного відновлення ключа генератора гами при $s = 2$ потрібно значно менше векторів ініціалізації в порівнянні з теоретичною верхньою межею (2.16). Навпаки, при $s = 10$ значення r_{pr} є помітно більше ніж r. Цю особливість можна пояснити тим, що в останньому випадку порушується припущення (2.9). Як наслідок, за умови $r' \le r$ на другому етапі алгоритму 2.1 метод максимуму правдоподібності приймає деякий помилковий вектор за істинний (тобто знаходить вектор $y \ne MA_i k$ з тим самим або меншим значенням параметра (2.6) в порівнянні з вектором $MA_i k$). Наведений ефект поступово нівелюється зі збільшенням значення r' впритул до r_{pr}.

Таким чином, запропонована та викладена у підрозділі атака узагальнює ряд раніше відомих [9, 10, 25 – 28] та може бути практично застосовною до більш широкого класу генераторів гами за менш жорстких обмежень відносно їх функцій ускладнення. Як показують чисельні розрахунки, при $\varepsilon \ge 0{,}05$ та $s \le 10$ ця атака дозволяє відновлювати ключ довжини $l_0 = 128$ біт зі складністю не більше 2^{38} елементарних операцій, використовуючи при цьому не більше ніж 2^{15} відрізків гами (117 знаків кожен) разом з відповідними векторами ініціалізації.

2.2. Узагальнена статистична атака на синхронні потокові шифри

2.2.1. Наукові основи та алгоритм реалізації узагальненої статистичної атаки.

В попередньому підрозділі запропоновано атаку на

генератор гами з лінійним законом реініціалізації початкового стану та близькою до алгебраїчно виродженої функцією ускладнення. Проте у складі сучасних синхронних потокових шифрів використовуються більш складні генератори гами з нелінійним законом реініціалізації. Задача відновлення ключів таких шифрів якісно відрізняється від попередньої та є суттєво більш складною.

Даний підрозділ присвячено побудові узагальненої атаки на синхронні потокові шифри, що за певних умов є застосовною до будь-якого криптографічного алгоритму, який описується за допомогою булевої функції $F:\{0,1\}^{l_0}\times\{0,1\}^{l_1}\to\{0,1\}$, один з аргументів якої є секретним, а інший – загальнодоступним параметром. У випадку синхронного потокового шифру в якості функції F можна взяти знак вихідної послідовності генератора гами шифру в деякому такті. Ця функція може бути доступна противнику, як "чорна скриня" (зокрема, алгоритм, що реалізує таку функцію може бути невідомим). Запропонована атака є узагальненням атаки FKM [10], а також кубічної атаки [9].

Розглянемо синхронний потоковий шифр, який складається з генератора гами та алгоритму формування його початкового стану за ключем та вектором ініціалізації. Генератор гами являє собою скінченний автономний автомат з множиною станів V_N, функцією переходів $h:V_N\to V_N$ та функцією виходів $f:V_N\to\{0,1\}$, а алгоритм формування початкового стану задається відображенням $H:V_{l_0}\times V_{l_1}\to V_N$, де l_0 – довжина ключа, l_1 – довжина вектора ініціалізації. Знак гами в i-му такті, який відповідає ключу $k\in V_{l_0}$ та вектору ініціалізації $c\in V_{l_1}$, визначається за формулою

$$\gamma_i(k,c)=f(h^i(H(k,c))), \tag{2.20}$$

де h^i є i-м степенем відображення h відносно операції композиції, $i=0,1,\ldots$.

Вважається, що противник може на свій розсуд вибирати вектори ініціалізації та обчислювати при фіксованому, але невідомому йому ключі відповідні їм знаки гами, намагаючись при цьому відновити шуканий ключ. Вважається також, що противник володіє алгоритмом $\mathcal{A}$ опробування ключів, який дозволяє безпомилково знаходити ключ, якщо він міститься серед ключів, які перебираються (іншими словами, алгоритм повного перебору ключів характеризується стовідсотковою надійністю).

Для проведення атаки противник вибирає тим чи іншим чином функції-оракули $F: V_{l_0} \times V_{l_1} \to \{0,1\}$, $\varphi: V_{s+l_1} \to \{0,1\}$ та матрицю $M_0 \in \mathbf{F}_2^{l_0 \times s}$ рангу $s < l_0$, які задовольняють таким умовам:

а) існує ефективний алгоритм обчислення значень $F_k(c) = F(k,c)$, $c \in V_{l_1}$, яким би ні був невідомий фіксований ключ $k \in V_{l_0}$;

б) справедлива нерівність

$$\mathbf{P}_{k,c}\{F(k,c) = \varphi(kM_0, c)\} \geq 1 - \vartheta, \tag{2.21}$$

де k та c є незалежними випадковими векторами, перший з яких розподілений рівномірно на множині V_{l_0}, а другий – відповідно до деякого (не обов'язково рівномірного) закону $\mathbf{P}_c$ на множині V_{l_1}, $\vartheta \in (0, 1/2)$.

Відзначимо, що в якості функції F, яка задовольняє умові а), можна взяти відображення γ_i, яке ставить у відповідність парі $(k,c) \in V_{l_0} \times V_{l_1}$ знак гами (2.20) в деякому фіксованому такті $i = 0, 1, \ldots$. Можна також покласти $F_k(c)$ рівним значенню похідної відображення γ_i за напрямом деякого підпростору $L \subseteq V_{l_1}$ (невеликої вимірності):

$$F_k(c) = \bigoplus_{u \in L} \gamma_i(k, c \oplus u),\ (k,c) \in V_{l_0} \times V_{l_1}. \tag{2.22}$$

Якщо $u_1, \dots, u_l$ є довільним базисом підпростору L, то значення правої частини рівності (2.22) дорівнює $D_{u_l} \cdots D_{u_1} \gamma_i(k, c)$, де $D_u g(x) = g(x \oplus u) \oplus g(x)$, $x \in V_m$ є похідною функції $g : V_m \to \{0,1\}$ за напрямом u (див., наприклад, [86], с. 89). Зокрема, вибираючи в якості L підпростір, породжений векторами $e_{j_1}, \dots, e_{j_l}$, де $1 \le j_1 < \dots < j_l \le l_1$ (а e_j позначає двійковий вектор довжини l_1, всі координати якого, за виключенням j-ї, дорівнюють нулю, $j \in \overline{1, l_1}$), можна задати функцію F_k за формулою:

$$F_k(c) = D_{j_1} \cdots D_{j_l} \gamma_i(k, c),\ (k, c) \in V_{l_0} \times V_{l_1}. \qquad (2.23)$$

Саме такий спосіб визначення функцій використовується в [10], причому параметр l, який характеризує складність обчислень їх значень, зазвичай не перевищує 20.

На рис. 2.2 наведено алгоритм відновлення ключа синхронного потокового шифру за виконанням умов а) та б), який базується на результатах попереднього підрозділу а також ідеях роботи [10] (алгоритм 2.2).

Позначимо $T_{\mathcal{A}}$ часову складність алгоритму $\mathcal{A}$, T_F та T_φ – часові складності обчислення значень F та φ відповідно. Тоді часова складність першого етапу узагальненого алгоритму відновлення ключа складає $O\left(2^s r(T_F + T_\varphi)\right)$ операцій.

Оскільки $\operatorname{rank}(M_0) = s$, то система рівнянь (2.25) має $2^{l_0 - s}$ розв'язків, перебір яких з метою знаходження ключа k потребує $O\left(s(l_0 - s)2^{l_0 - s} T_{\mathcal{A}}\right)$ операцій. Таким чином, часова складність узагальненого алгоритму відновлення ключа складає

$$T(l_0, s, r) = O\left(2^s r(T_F + T_\varphi) + s(l_0 - s)2^{l_0 - s} T_{\mathcal{A}}\right) \qquad (2.24)$$

операцій.

Алгоритм 2.2

Вхідні дані:

– алгоритм $\mathcal{A}$ опробування ключів, який має стовідсоткову надійність;

– функції F_k та φ, задані за допомогою оракулів (де k є невідомим ключем);

– матриця M_0.

Параметр: $r \in \mathbf{N}$.

Етап 1 (*відновлення вектора* kM_0 *методом максимальної правдоподібності*):

– згенерувати незалежні випадкові вектори ініціалізації $c^{(1)}, \ldots, c^{(r)}$, що розподілені на множині V_{l_1} за законом P_c;

– для кожного $y \in V_s$ обчислити значення

$$\nu(y) = \sum_{j=1}^{r} (\varphi(y, c^{(j)}) \oplus F_k(c^{(j)}))$$

та знайти вектор $\hat{y} \in V_s$ такий, що $\nu(\hat{y}) = \min\limits_{y \in V_s} \nu(y)$.

Етап 2 (*відновлення ключа*). Перебираючи всі розв'язки $\tilde{k} \in V_{l_0}$ системи лінійних рівнянь

$$\tilde{k} M_0 = \hat{y}, \qquad (2.25)$$

знайти шуканий ключ за допомогою алгоритму $\mathcal{A}$.

Рис. 2.4. Алгоритм реалізації узагальненої статистичної атаки

Для того щоб оцінити обсяг матеріалу r, для якого алгоритм 2.2 дозволяє знаходити ключ із заданою достовірністю, зробимо одне додаткове

припущення.

Зафіксуємо число $\varepsilon \in (0,1)$. Назвемо ключ $k \in V_{l_0}$ ε-*слабким* (*відносно описаної атаки*), якщо $\mathbf{P}_c\{F_k(c) = \varphi(kM_0, c)\} \geq 1/2\,(1+\varepsilon)$ та припустимо, що, поряд з умовами а), б) виконується наступна умова:

в) для будь-якого ε-слабкого ключа $k \in V_{l_0}$ та довільного вектора $y \in V_s \setminus \{kM_0\}$ справедлива рівність

$$\mathbf{P}_c\{F_k(c) = \varphi(y, c)\} = 1/2\,. \tag{2.26}$$

Наступна лема доводиться аналогічно лемі 2.1.

Лема 2.2. Нехай k є ε-слабким ключем, $\varepsilon \in (0,1)$. Тоді за умов б), в) ймовірність правильного відновлення вектора kM_0 на першому етапі алгоритму 2.2 обмежена знизу величиною $1 - 2^s \exp\{-1/8 \cdot r\,\varepsilon^2\}$.

Лема 2.3. За умови б) для будь-якого $\varepsilon \in (0,1)$ існує не менше ніж $2^{l_0}\left(1 - \dfrac{2\vartheta}{1-\varepsilon}\right)$ ε-слабких ключів.

Доведення. Позначимо $p_k = \mathbf{P}_c\{F_k(c) = \varphi(kM_0, c)\}$, $k \in V_{l_0}$. Тоді ймовірність в лівій частині рівності (2.21) дорівнює $2^{-l_0} \sum\limits_{k \in V_{l_0}} p_k \geq 1 - \vartheta$. При цьому ключ k не є ε-слабким тоді й тільки тоді, коли $p_k < 1/2\,(1+\varepsilon)$. Отже, ймовірність того, що випадково рівноймовірно обраний ключ не виявиться ε-слабким дорівнює

$$\mathbf{P}_k\{p_k < 1/2\,(1+\varepsilon)\} = \mathbf{P}_k\{1 - p_k > 1/2\,(1-\varepsilon)\} \leq \frac{2}{1-\varepsilon}\, 2^{-l_0} \sum_{k \in V_{l_0}} (1 - p_k) \leq \frac{2\vartheta}{1-\varepsilon}\,.$$

Звідси знаходимо оцінку числа ε-слабких ключів:

$$2^{l_0}(1-\mathbf{P}_k\{p_k<1/2(1+\varepsilon)\})\geq 2^{l_0}\left(1-\frac{2\vartheta}{1-\varepsilon}\right).$$

Лему доведено.

На підставі лем 2.2, 2.3 справедлива така теорема, яка дозволяє оцінити ефективність запропонованої атаки.

Теорема 2.2. Нехай виконуються умови а), б), в) при $\vartheta=1/2-2^{-\mu}$, $\varepsilon=(2^{\mu}-1)^{-1}$, де $\mu>1$. Покладемо $r=\left\lceil 2^{2\mu+3}\ln(2^{s}\delta^{-1})\right\rceil$, $\delta\in(0,1)$. Тоді існує не менше ніж $2^{l_0-\mu}$ ε-слабких ключів, кожен з яких може бути відновлений за допомогою алгоритму 2.2 з ймовірністю $1-\delta$ зі складністю, яка визначається за формулою (2.24).

Відзначимо такі відмінності викладених вище результатів від результатів роботи [10]:

– запропоновано статистичну атаку на синхронні потокові шифри, яка базується на застосуванні суттєво більш широкого класу наближень функції F (а саме, алгебраїчно вироджених функцій виду $\varphi(kM_0, c)$, $k\in V_{l_0}$, $c\in V_{l_1}$);

– охарактеризований клас ключів (ε-слабкі ключі), які можуть бути відновлені за допомогою описаної атаки, та отримано нижню оцінку кількості ключів в цьому класі;

– наведено явний вираз трудомісткості атаки як функції її надійності, складності обчислення значень функцій-оракулів та алгоритму опробування ключів;

Відзначимо також, що у випадку, коли функція F_k визначається за формулою (2.23), функція φ є афінною, а параметр ϑ у формулі (2.21) дорівнює нулю, описана статистична атака зводиться до алгебраїчної (кубічної) атаки, запропонованої в [9].

2.2.2. Модифікація алгоритму FKM побудови наближень булевих функцій для запропонованої атаки. Згідно з формулою (2.24) для застосування та оцінки трудомісткості узагальненої статистичної атаки необхідно деяким чином задати функцію φ, за якою визначається алгебраїчно вироджене наближення вхідної функції F. В [10] запропоновано простий детермінований алгоритм задання функції φ, який полягає у фіксації певних змінних функції F нульовими значеннями. На прикладах шифрів Grain-128 та Trivium показано, що цей алгоритм дозволяє будувати прийнятні наближення вхідних функцій оракулів, але не проведено аналізу ефективності цього алгоритму в загальному випадку.

Нижче пропонується іншій, ймовірнісний алгоритм побудови функції φ, який є модифікацією алгоритму з [10]. Отримано аналітичну верхню межу середньої відстані між функцією F та її наближенням, що будується за допомогою запропонованого алгоритму.

Для будь-яких $f, g, F \in B_n$ позначимо

$$d(f,g) = \mathbf{P}_X\{f(X) \neq g(X)\} = 2^{-n}\#\{x \in V_n : f(x) \neq g(x)\},$$

$$\hat{F}(\alpha) = 2^{-n}\sum_{x \in V_n}(-1)^{F(x) \oplus \alpha x},\ \alpha \in V_n, \tag{2.27}$$

$$D_\alpha F(x) = F(x \oplus \alpha) \oplus F(x),\ x \in V_n. \tag{2.28}$$

Для будь-якого $m \in \overline{0,n}$ позначимо $L_{n,m}$ множину всіх m-вимірних підпросторів векторного простору V_n. Покладемо

$$\omega_F(H) = \sum_{\beta \in H}\hat{F}(\beta)^2,\ H \in L_{n,m}. \tag{2.29}$$

Справедлива рівність (див., наприклад, теорему 2.89 в [86])

$$\omega_F(H) = 1 - 2 \cdot \left(2^{-(n-m)} \sum_{\alpha \in H^{\perp}} wt(D_\alpha F) \right),\ H \in L_{n,m}. \tag{2.30}$$

Доведемо твердження, яке складає теоретичну основу алгоритму, що пропонується, побудови функції φ.

Твердження 2.2. Нехай $F \in B_n$, $M \in \mathrm{F}_2^{n \times m}$, $\mathrm{rank}(M) = m \in \overline{1, n-1}$, H – підпростір, породжений стовпцями матриці M і U – оборотна $n \times n$-матриця така, що $UM = \begin{pmatrix} I_m \\ 0 \end{pmatrix}$. Для будь-якого $z \in V_{n-m}$ покладемо

$$\varphi_z(y) = F((y, z)U),\ y \in V_m. \tag{2.31}$$

Тоді при випадковому рівноймовірному виборі вектора z середнє значення відносної відстані між функціями F та $g_z(x) = \varphi_z(xM)$, $y \in V_m$ не перевищує $1 - \omega_F(H)$, де $\omega_F(H)$ визначається за формулою (2.29) . Зокрема, за умови $\omega_F(H) \geq \theta$ справедлива нерівність $\mathbf{E}_z d(F, g_z) \leq 1 - \theta$.

Доведення. Згідно з означенням функції g_z

$$\mathbf{E}_z d(F, g_z) = 2^{-(n-m)} \sum_{z \in V_{n-m}} 2^{-n} \sum_{x \in V_n} (F((xM, z)U) \oplus F(x)) =$$

$$= 2^{-(n-m)} \sum_{z \in V_{n-m}} 2^{-n} \sum_{x \in V_n} (F((xUM, z)U) \oplus F(xU)) =$$

$$2^{-(n-m)} \sum_{z \in V_{n-m}} 2^{-n} \sum_{x \in V_n} (F((x \begin{pmatrix} I_m \\ 0 \end{pmatrix}, z)U) \oplus F(xU)) =$$

$$= 2^{-(n-m)} \sum_{z \in V_{n-m}} 2^{-n} \sum_{\substack{x_1 \in V_m, \\ x_2 \in V_{n-m}}} (F((x_1, z)U) \oplus F((x_1, x_2)U)) \le a + b,$$

де

$$a = 2^{-n}\ 2^{-(n-m)} \sum_{\substack{z \in V_{n-m}, \\ x_1 \in V_m,\ x_2 \in V_{n-m}}} (F((x_1, z)U) \oplus F((x_1,\ z \oplus x_2)U)),$$

$$b = 2^{-n}\ 2^{-(n-m)} \sum_{\substack{z \in V_{n-m}, \\ x_1 \in V_m,\ x_2 \in V_{n-m}}} (F((x_1, z \oplus x_2)U) \oplus F((x_1,\ x_2)U)).$$

Далі,

$$a = 2^{-n}\ 2^{-(n-m)} \sum_{\substack{z \in V_{n-m}, \\ x_1 \in V_m,\ x_2 \in V_{n-m}}} (F((x_1, z)U) \oplus F((x_1, z)U \oplus (0, x_2)U)) =$$

$$= 2^{-(n-m)}\ 2^{-n} \sum_{\substack{z \in V_{n-m}, \\ x_1 \in V_m,\ x_2 \in V_{n-m}}} D_{(0,x_2)U} F((x_1, z)U) = 2^{-(n-m)} \sum_{x_2 \in V_{n-m}} wt(D_{(0,x_2)U} F),$$

$$b = 2^{-n}\ 2^{-(n-m)} \sum_{\substack{z \in V_{n-m}, \\ x_1 \in V_m,\ x_2 \in V_{n-m}}} (F((x_1, x_2)U) \oplus F((x_1, x_2)U \oplus (0, z)U)) =$$

$$= 2^{-(n-m)}\ 2^{-n} \sum_{\substack{z \in V_{n-m}, \\ x_1 \in V_m,\ x_2 \in V_{n-m}}} D_{(0,z)U} F((x_1, x_2)U) = 2^{-(n-m)} \sum_{z \in V_{n-m}} wt(D_{(0,z)U} F).$$

Отже,

$$\mathbf{E}_z d(F, g_z) \le 2 \cdot 2^{-(n-m)} \sum_{x_2 \in V_{n-m}} wt(DF_{(0,x_2)U}) = 2 \cdot 2^{-(n-m)} \sum_{\alpha \in H^{\perp}} wt(D_\alpha F),$$

де остання рівність випливає з означень підпростору H та матриці U. Нарешті, використовуючи формулу (2.30), отримаємо, що $\mathbf{E}_z d(F, g_z) \le 1 - \omega_F(H)$.

Твердження доведено.

Отже, наведемо алгоритм обчислення значень функції $\varphi \in B_m$ (рис. 2.5), який базується на розвитку ідеї [10] використання в ролі наближень функції F її підфункцій і твердженні 2.2.

Алгоритм 2.3

Вхідні дані:

– функція $F \in B_n$, задана за допомогою оракула ($n = l_0 + l_1$);

– матриця $M_0 \in \mathbf{F}_2^{l_0 \times s}$ де $\operatorname{rank}(M_0) = s$ ($s \in \overline{1, l_0 - 3}$).

Етап 1 (*попередні обчислення*):

– покласти $m = s + l_1$, $M = \begin{pmatrix} M_0 & 0 \\ 0 & I_{l_1} \end{pmatrix}$;

– обчислити оборотну матрицю $U \in \mathrm{F}_2^{n \times n}$ таку, що $UM = \begin{pmatrix} I_m \\ 0 \end{pmatrix}$, використовуючи метод Гауса;

– згенерувати випадковий рівноймовірний вектор $z \in V_{n-m}$.

Етап 2: Для будь-якого $y \in V_m$ покласти $\varphi(y) = F((y, z)U)$, $y \in V_m$.

Рис. 2.5. Ймовірнісний алгоритм обчислення значень функції φ

Зауважимо, що формування матриці M, обчислення матриці U та генерування випадкового вектора z на першому етапі алгоритму 2.3 здійснюються одноразово, тобто зазначені об’єкти не змінюються зі зміною аргументу y функції φ.

На підставі твердження 2.2 середня (за всіма $z \in V_{n-m}$) відносна відстань між функцію F та її наближенням вигляду $\varphi(xM)$, $x \in V_n$ не перевищує $1 - \sum_{y \in V_m} \hat{F}(My)^2$. При цьому часова складність обчислення значення функції φ є $T_\varphi = O(T_F n^2)$, де T_F – складність обчислення значення функції F.

2.2.3. Приклад практичного застосування запропонованої атаки. Застосуємо отримані наукові результати до побудови статистичної атаки на редуковану версію шифру Grain-128 (рис. 2.6). Нагадаємо, що цей шифр є одним з трьох потокових шифрів, орієнтованих на апаратну реалізацію, що рекомендовані за підсумками дослідного проекту eSTREAM. Шифр має довжину ключа $l_0 = 128$ біт і довжину вектора ініціалізації $l_1 = 96$ біт. При цьому алгоритм формування початкового стану генератора гами шифру являє собою ітераційну процедуру, яка складається з 256 однотипних кроків (раундів).

У [10] (приклад 9) описано атаку на редуковану версію Grain-128, де замість 256 раундів алгоритму формування початкового стану виконується тільки 180. Для побудови цієї атаки використовується певна функція $F = F_{\mathrm{FKM}}(k, c)$, де $k = (k_0, \ldots, k_{127}) \in V_{128}$, $c = (c_0, \ldots, c_{95}) \in V_{96}$, а також її наближення, яке отримується шляхом фіксації певних її 18 змінних k_i, $i \in \overline{0,127}$, нульовими значеннями.

Для побудови більш ефективної атаки знайдемо кращі наближення функції F за допомогою алгоритму 2.3, використовуючи 128×52-матрицю M_0, наведену на рис. 2.7 (загальний метод знаходження таких матриць для побудови алгебраїчно вироджених наближень булевих функцій викладено в розділі 4).

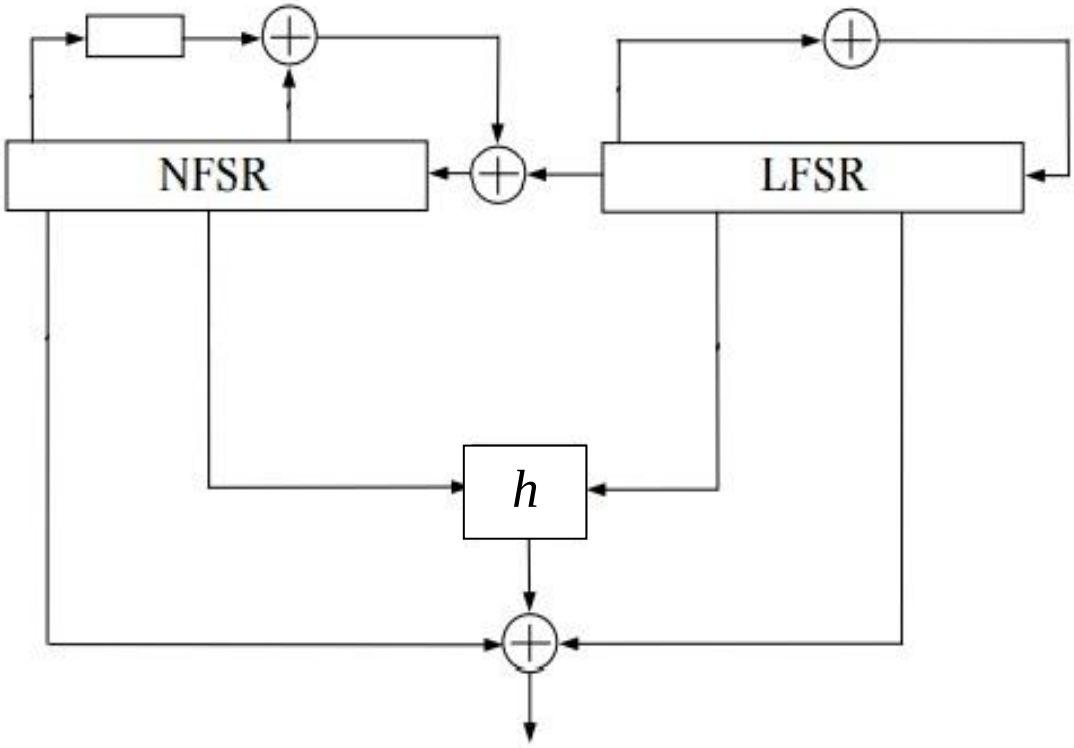

Рис.2.6. Схема генератора гами шифру Grain-128

В табл. 2.5 наведено результати застосування алгоритму 2.3 для побудови наближень функції $F = F_{\mathrm{FKM}}(k, c)$. Для оцінювання відносної відстані d між функціями F та $g(k, c) = \varphi(kM, c)$, $k \in V_{l_0}$, $c \in V_{l_1}$ з точністю $\varepsilon \in (0, 1)$ та достовірністю $1 - \delta$ використано стандартний алгоритм (див., наприклад, [23]):

– покласти $t = \left\lceil 2^{-1} \varepsilon^{-2} \ln(\delta^{-1}) \right\rceil$;

– згенерувати незалежні випадкові вектори $X_1, \ldots, X_t$ з рівномірним розподілом на множині V_n;

– обчислити значення $d = t^{-1} \sum_{i=1}^{t} (F(X_i) \oplus g(X_i))$.

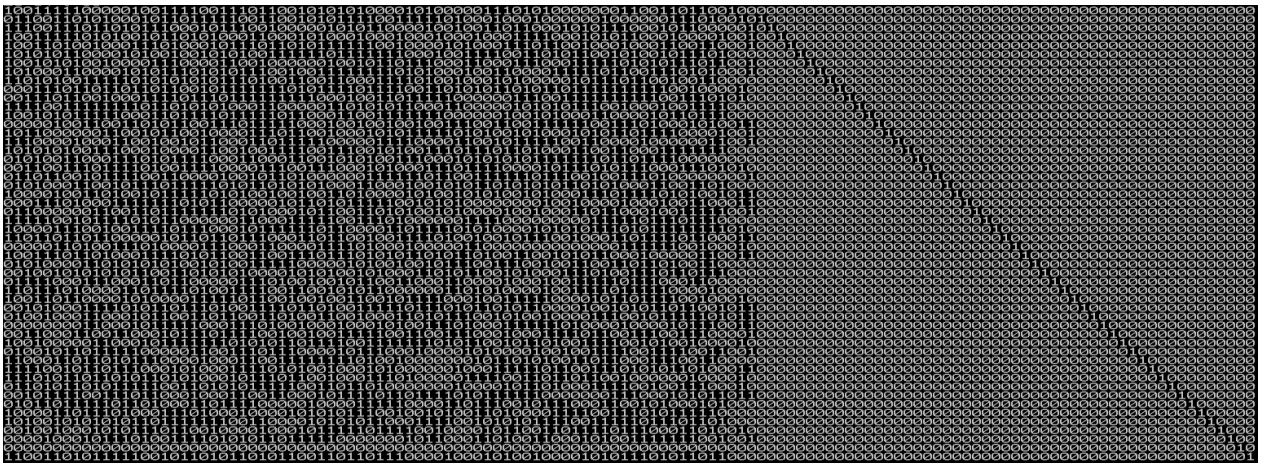

Рис. 2.7. Матриця, транспонована до M_0

Таблиця 2.5

Результати застосування алгоритму 2.3 ($s = 52$, $\varepsilon = 0{,}05$, $\delta = 0{,}1$)

d	$d + \varepsilon$	T, хв.
0,1876	0,1926	40
0,1652	0,1702	40
0,2103	0,2153	40
0,3286	0,3336	40
0,2056	0,2106	40
0,2762	0,2812	40
0,2634	0,2684	40
0,1844	0,1894	40
0,1786	0,1836	40
0,1779	0,1829	40

Як видно з табл. 2.5, найменше (з десяти отриманих) значень верхніх оцінок параметра $d(F, g)$ дорівнює 0,1702. Таким чином, $d(F, g) \le 0{,}1702$ з достовірністю не менше ніж $1 - \delta = 0{,}9$. При цьому двійкове значення вектора z, який генерується на першому етапі алгоритму 2.3 та визначає наближення $g(k, c) = \varphi(kM, c)$, є таким:

1101011101111110000000111001010111111000000011000110010010001010110010010100

Використовуючи отримане наближення функції F, можна побудувати узагальнену статистичну атаку на редуковану версію шифру Grain-128.

В табл. 2.6 наведено залежності трудомісткості узагальненої статистичної атаки на зазначену версію шифру Grain-128 від параметрів s і ϑ наближень, які будуються за допомогою алгоритму 2.3 (див. формулу (2.21)). Як видно з таблиці, трудомісткість атаки суттєво залежить від значення s (довжини вектора kM_0, який відновлюється на першому кроці алгоритму). Найменше значення трудомісткості досягається при $s = 61$ та складає від 2^{89} до 2^{98} операцій в залежності від значення ϑ.

Таблиця 2.6

Оцінки трудомісткості узагальненої статистичної атаки

ϑ	s	$T(l_0, s, r)$
0,49	30	2^{119}
	40	2^{109}
	50	2^{99}
	60	2^{98}
	70	2^{108}
0,35	30	2^{119}
	40	2^{109}
	50	2^{99}
	60	2^{91}
	70	2^{108}
0,20	30	2^{119}
	40	2^{109}
	50	2^{99}
	60	2^{90}
	70	2^{98}
0,01	30	2^{119}
	40	2^{109}
	50	2^{99}
	60	2^{89}
	70	2^{97}

Нагадаємо, що на підставі теореми 2.2 трудомісткість запропонованої атаки визначається за формулою (2.24), де $r = \left\lceil 2^{2\mu+3} \ln (2^s \delta^{-1}) \right\rceil$, а число $\mu > 1$ визначається з нерівності $d(f,g) \le 1/2 - 2^{-\mu}$. Вважаючи $T_F = 128$, $T_\varphi = T_F n^2$, $T_{\mathcal{A}} = 3 \cdot 128$, та використовуючи знайдену оцінку $d(F,g) \le 0{,}1702$, отримаємо, що (за умови теореми 2.2) трудомісткість узагальненої статистичної атаки на зазначену версію шифру Grain-128 не перевищує 2^{97}, що є в 2^{27} разів менше ніж трудомісткість раніше відомої атаки [10].

Отже, наведений приклад демонструє суттєво більшу ефективність запропонованої статистичної атаки в порівнянні з раніше відомою [10].

Висновки

1. Основним науковим результатом розділу є статистичні атаки на синхронні потокові шифри, які узагальнюють низку раніше відомих атак [9, 10, 25 – 28], надаючи криптоаналітику більше можливостей для вибору як функцій-оракулів, так і їх наближень. Перша атака спрямована на відновлення ключів генераторів гами з лінійним законом реініціалізації початкового стану і узагальнює атаки, наведені в [25 – 28]. Друга атака є узагальненням більш потужної атаки FKM [10], а також кубічної атаки [9] і базується на наближенні булевих функцій, що реалізуються алгоритмами шифрування в цілому, алгебраїчно виродженими функціями.

2. Необхідною умовою застосовності запропонованих атак є існування для заданої функції-оракула $F = F(k,c)$, яка визначається алгоритмом шифрування, алгебраїчно виродженого наближення вигляду $g(k,c) = \varphi(kM_0, c)$, $k \in V_{l_0}$, $c \in V_{l_1}$, де M_0 є $l_0 \times s$-матрицею рангу s, φ – булевою функцією від $s + l_1$ змінних, $s < n = l_0 + l_1$. Показано, що трудомісткості обох атак залежать

експоненційно від параметра s і квадратично від параметра $\varepsilon^{-1} = (1-2d(F,g))^{-1}$, де $d(F,g)$ є відносною відстанню між функціями F та g. Крім того, трудомісткість другої атаки залежить лінійно від складності обчислення значення функції φ та алгоритму опробування ключів, що використовується (див. формули (2.16), (2.17), (2.24) і теорему 2.2).

3. Результати моделювання першої з двох запропонованих атак показують, що за певних умов вона може бути успішно застосована на практиці до генераторів гами з довжиною ключа, принаймні, 128 біт. Зокрема, при $\varepsilon \geq 0{,}05$ та $s \leq 10$ зазначена атака дозволяє відновлювати ключ такої довжини зі складністю не більше 2^{38} елементарних операцій, використовуючи не більше 2^{15} відрізків гами (довжиною не більше 117 знаків кожен) разом з відповідними векторами ініціалізації.

4. Окремим науковим результатом розділу є запропонований алгоритм 2.3 побудови булевих функцій від меншої кількості змінних, які використовуються для визначення наближень при проведенні узагальненої статистичної атаки. Цей алгоритм є модифікацією (детермінованого) алгоритму з [10], проте, на відміну від останнього, має теоретичне обґрунтування. Показано, що алгоритм 2.3 дозволяє будувати "якісні" наближення з погляду відносної відстані до функції-оракула F (твердження 2.2). При цьому часова складність алгоритму дорівнює $O(T_F n^2)$, де T_F є складністю обчислення значення функції F, а n є кількістю її змінних.

5. Застосування отриманих результатів до редукованої версії шифру Grain-128 показує, що трудомісткість узагальненої статистичної атаки на цю версію шифру не перевищує 2^{97}, що є в 2^{27} разів менше ніж трудомісткість раніше відомої атаки [10].

Основні наукові результати розділу опубліковані в [30, 32, 34, 36, 38 – 40, 42, 43].

РОЗДІЛ 3

МЕТОДИ ПОБУДОВИ ТА ОБГРУНТУВАННЯ ВІДСУТНОСТІ ВИСОКОЙМОВІРНИХ *k*-ВИМІРНИХ НАБЛИЖЕНЬ БУЛЕВИХ ФУНКЦІЙ, ЩО ВИКОРИСТОВУЮТЬСЯ У СИНХРОННИХ ПОТОКОВИХ ШИФРАХ

В попередньому розділі викладно статистичні атаки на синхронні потокові шифри, які узагальнюють низку раніше відомих атак [9, 10, 25 – 28]. Показано, що необхідною умовою стійкості цих шифрів відносно зазначених атак є неможливість наближення функцій ускладнення, які використовуються в їх конструкціях, k-вимірними функціями, тобто булевими функціями від n змінних, що є лінійно еквівалентними функціям від суттєво меншого числа k змінних.

В розділі 1 зазначено, що ефективність розв'язання задач знаходження та обґрунтування відсутності k-вимірних наближень булевої функції f суттєво залежить від відносної відстані між нею та її шуканими наближеннями. Якщо ця відстань не перевищує $2^{-(k+1)}(1-\varepsilon)$, де $\varepsilon \in (0,1)$, то існує не більше однієї шуканої функції, для знаходження якої відомі ефективні алгоритми [23]. Суттєво більш складною задачею є побудова списку всіх високоймовірних наближень функції f від n змінних, тобто k-вимірних функцій степеня не вище d, які знаходяться від f на відносній відстані не більше $2^{-d}(1-\varepsilon)$, $1 \le d \le k < n$, $\varepsilon \in (0,1)$. Найефективніший з відомих алгоритмів розв'язання цієї задачі запропоновано П. Гопаланом [24].

В даному розділі викладено метод побудови високоймовірних наближень булевих функцій, який суттєво покращує алгоритм Гопалана (має меншу трудомісткість в порівнянні з останнім, у певних випадках – в 1000 та більше разів). Запропонований метод базується на окремих результатах статті [143], які

дозволяють встановити більш точну (в порівнянні з [24]) оцінку кількості шуканих наближень, а також на детальному аналізі структури таких наближень, що надає можливість помітно скоротити часову складність їх побудови. Показано, що цей метод може бути застосований на практиці при аналізі кореляційних властивостей функцій ускладнення синхронних потокових шифрів при малих значеннях k і d, якщо кількість "відносно великих" за модулем коефіцієнтів Уолша-Адамара функції f не перевищує 20.

Іншим науковим результатом розділу є метод обґрунтування відсутності високоймовірних наближень булевих функцій, який базується на ймовірнісному алгоритмі обчислення значень нижніх меж відносної відстані між зрівноваженою булевою функцією та множиною всіх k-вимірних функцій від n змінних. На відміну від відомих (тривіальних детермінованих) алгоритмів розв'язання цієї задачі, складність запропонованого алгоритму залежить лінійно від n та поліноміально від величин, обернених до точності та імовірності помилки алгоритму. Наведено результати моделювання цього алгоритму, які дозволяють стверджувати про можливість його застосування на практиці при обґрунтуванні стійкості функцій ускладнення синхронних потокових шифрів відносно статистичних атак, наведених в розділі 2 .

3.1. Метод побудови високоймовірних k-вимірних наближень булевих функцій

3.1.1. Постановка задачі, теоретичні результати та базові алгоритми.

Нижче використовуються такі позначення:

V_n – векторний простір двійкових векторів довжини n;

$F_{n\times k}$ – множина матриць розміру $n\times k$ над полем $F = \mathrm{GF}(2)$;

$C(A)$ – підпростір векторного простору V_n, породжений стовпцями матриці $A \in F_{n\times k}$;

B_n – множина булевих функцій від n змінних;

$\deg g$ – степінь поліному Жегалкіна функції $g \in B_n$;

$d(f,g) = 2^{-n} |\{x \in V_n : f(x) \neq g(x)\}|$ – відносна відстань між функціями $f, g \in B_n$;

$\hat{f}(\alpha) = 2^{-n} \sum_{x \in V_n} (-1)^{f(x) \oplus \alpha x}$, $\alpha \in V_n$ – нормовані коефіцієнти Уолша-Адамара функції $f \in B_n$.

Для будь-якої функції $g \in B_n$ покладемо $I_g = \{\alpha \in V_n \mid \forall x \in V_n : g(x \oplus \alpha) = g(x)\}$ та позначимо $I_g{}^{\perp}$ підпростір, дуальний до векторного простору I_g над полем F (див., наприклад, [86]).

Нагадаємо (див. підрозділ 1.3.2), що функція g називається k-вимірною, якщо вона може бути представлена у вигляді

$$g(x) = \varphi(xA), \; x \in V_n, \tag{3.1}$$

де $\varphi \in B_k$, $A \in F_{n\times k}$, та строго $\boldsymbol{k}$-вимірною, якщо k є найменшим невід'ємним цілим числом, для якого існує представлення функції g у вигляді (3.1). Кожне таке представлення, що відповідає найменшому можливому значенню $k \in \overline{0,n}$, називається незвідним представленням функції g [29, 143].

Позначимо $B_{n,k}$ множину k-вимірних функцій від n змінних, $\overline{B}_{n,k} = B_{n,k} \setminus B_{n,k-1}$. Для будь-яких $f \in B_n$, $\varepsilon \in (0,1)$, $d, k \in \mathrm{N}$, де $d \le k < n$, покладемо

$$B_{n,k,d}(f;\varepsilon) = \{g \in B_{n,k} : d(f,g) \le 2^{-d}(1-\varepsilon), \deg g \le d\}, \tag{3.2}$$

$$\overline{B}_{n,k,d}(f;\varepsilon)=\{g\in\overline{B}_{n,k}:d(f,g)\le 2^{-d}(1-\varepsilon),\deg g\le d\}. \qquad (3.3)$$

Потрібно розробити метод побудови множини (3.2) за вектором значень функції f та числами d, k і ε.

Повне вирішення цієї задачі викладено в наступному пункті розділу. Даний пункт присвячено розв'язанню окремої підзадачі, яка полягає в розробці алгоритмів побудування множини (3.3) для випадків, коли $d<k$ та $d=k$ відповідно. Алгоритми, що пропонуються, базуються на низці тверджень, які наведені нижче.

Перше з них є основним та випливає з теореми 4 і леми 5 у [143].

Твердження 3.1. Нехай

$$\mu_0=\max\{2^{1-k}\varepsilon,\ \frac{4}{3\sqrt{3}}2^{-k/2-d/2}\varepsilon^{3/2}\},$$

$$S_f(\mu_0)=\{\alpha\in V_n:|\hat{f}(\alpha)|\ge\mu_0\}. \qquad (3.4)$$

Тоді кожна функція $g\in\overline{B}_{n,k,d}(f;\varepsilon)$ задовольняє умові

$$I_g{}^{\perp}=\left\langle\{x\in I_g{}^{\perp}:x\in S_f(\mu_0)\}\right\rangle; \qquad (3.5)$$

іншими словами, векторний простір $I_g{}^{\perp}$ породжується всіма векторами $x\in I_g{}^{\perp}$, які належать множині (3.4).

Наступне твердження є безпосереднім наслідком твердження 2 в [143] (див. також [29], наслідок 1).

Твердження 3.2. Представлення (3.1) є незвідним тоді й тільки тоді, коли $\text{rank}\,(A) = k$ та $I_\varphi = \{0\}$. При цьому $I_g = \{x \in V_n : xA = 0\} = C(A)^\perp$.

Твердження 3.3. Нехай $g \in \overline{B}_{n,k,d}(f;\varepsilon)$. Тоді функція g має незвідне представлення вигляду (3.1), в якому стовпці $\alpha_1, ..., \alpha_k$ матриці A належать множині (3.4).

Доведення. Зафіксуємо будь-яке незвідне представлення функції g: $g(x) = \varphi'(xA')$, $x \in V_n$, де $\varphi' \in B_k$, $A' \in F_{n\times k}$,

$$I_{\varphi'} = \{0\},\ \text{rank}(A') = k. \tag{3.6}$$

Згідно з твердженням 3.2, виконується рівність $I_g{}^\perp = C(A')$. З іншого боку, на підставі формули (3.5) векторний простір $I_g{}^\perp$ має базис $\alpha_1, ..., \alpha_k \in S_f(\mu_0)$.

Позначимо A матрицю, що складається з вектор-стовпців $\alpha_1, ..., \alpha_k$. Тоді $C(A) = C(A')$ і, отже, існує оборотна матриця $U \in F_{k\times k}$ така, що $A' = AU$. Покладемо $\varphi(y) = \varphi'(yU)$, $y \in V_k$; тоді

$$g(x) = \varphi'(xA') = \varphi'((xA)U) = \varphi(xA),\ x \in V_n,$$

причому зазначене представлення функції g є незвідним внаслідок рівностей (3.6) та означення функції φ.

Отже, твердження доведено.

Твердження 3.4. Для будь-якого незвідного представлення (3.1) функції $g \in \overline{B}_{n,k,d}(f;\varepsilon)$ виконується співвідношення $\deg\varphi = \deg g \le d$.

Доведення. Оскільки ранг матриці A у правій частині рівності (3.1) дорівнює k, існує оборотна матриця $W \in F_{n\times n}$ така, що $WA = \begin{pmatrix} E_k \\ 0_{n-k} \end{pmatrix}$, де E_k та 0_{n-k} – одинична та нульова матриці зазначених порядків.

Розглянемо функцію $g'(x) = g(xW)$, $x \in V_n$, лінійно еквівалентну функції g. Тоді $\deg g' = \deg g \le d$. З іншого боку, для будь-якого $x = (x_1, ..., x_n) \in V_n$ виконуються рівності

$$g'(x) = g(xW) = \varphi(xWA) = \varphi(x\begin{pmatrix} E_k \\ 0_{n-k} \end{pmatrix}) = \varphi(x_1, ..., x_k),$$

з яких випливає, що $\deg g' = \deg\varphi$. Отже, $\deg\varphi = \deg g' = \deg g \le d$.

Твердження доведено.

Занумеруємо зараз елементи множини $S_f(\mu_0) = \{\alpha_1, ..., \alpha_m\}$ таким чином, щоб $|\hat{f}(\alpha_1)| \ge \cdots \ge |\hat{f}(\alpha_m)|$; зауважимо, що $m \le (\mu_0)^{-2}$ (див., наприклад, [157], п. 3.2).

Позначимо $S_f(\mu_0)^{\langle k\rangle}$ сукупність усіх $n\times k$-матриць $A = (\alpha_{i_1}, ..., \alpha_{i_k})$, які складаються з лінійно незалежних вектор-стовпців $\alpha_{i_1}, ..., \alpha_{i_k} \in S_f(\mu_0)$, що задовольняють умові $1 \le i_1 < ... < i_k \le m$. Будь-які матриці $A, A' \in S_f(\mu_0)^{\langle k\rangle}$ вважатимемо еквівалентними, якщо множини їх стовпців породжують той самий підпростір векторного простору V_n, тобто існує оборотна матриця $U \in F_{k\times k}$ така, що $A' = AU$.

Твердження 3.5. Нехай $A_1, \dots, A_l$ – будь-яка система представників усіх класів еквівалентності матриць з множини $S_f(\mu_0)^{\langle k \rangle}$. Тоді кожна функція $g \in \overline{B}_{n,k,d}(f;\varepsilon)$ має єдине незвідне представлення вигляду $g(x) = \psi(xA_j)$, $x \in V_n$, де $j \in \overline{1,l}$. При цьому виконується нерівність $\deg\psi \le d$.

Доведення. На підставі твердження 3.3 існує незвідне представлення (3.1) функції g таке, що $A \in S_f(\mu_0)^{\langle k \rangle}$. Крім того, існує точно одна матриця A_j, $j \in \overline{1,l}$, еквівалентна матриці A.

Нехай $A = A_j U$, де U – оборотна матриця порядку k. Покладемо $\psi(y) = \varphi(yU)$, $y \in V_k$. Тоді $g(x) = \varphi(xA) = \varphi(xA_jU) = \psi(xA_j)$, $x \in V_n$, причому останнє представлення функції g є незвідним. Звідси на підставі твердження 3.4 отримаємо, що $\deg\psi \le d$.

Припустимо, що поряд із наведеним, існує ще одне незвідне представлення функції g того ж самого вигляду: $g(x) = \psi'(xA_{j'})$, $x \in V_n$, де $j' \in \overline{1,l}$. Тоді стовпці кожної з матриць A_j, $A_{j'}$ породжують той самий підпростір $I_g{}^{\perp}$ і, отже, $j = j'$. Нарешті, оскільки $\operatorname{rank}(A_j) = k$, то з рівностей $\psi(xA_j) = \psi'(xA_j)$, $x \in V_n$, випливає, що $\psi = \psi'$.

Таким чином, функція g має єдине незвідне представлення вигляду $g(x) = \psi(xA_j)$, $x \in V_n$, де $j \in \overline{1,l}$, причому $\deg\psi \le d$, що й треба було довести.

Останнє твердження дозволяє запропонувати алгоритм побудови множини (3.3) у випадку, коли $d < k$ (рис. 3.1).

Коректність цього алгоритму випливає безпосередньо з твердження 3.5.

Зауважимо, що для побудови системи $A_1, \dots, A_l$ та перевірки умови $I_\varphi = \{0\}$ $(\varphi \in B_k)$ на другому етапі алгоритму 3.1 потрібно виконати певні обчислення, обсяг яких може виявитися досить значним. Тому на цьому етапі

можна організувати перебір усіх матриць $A \in S_f(\mu_0)^{\langle k \rangle}$ та функцій $\varphi \in B_k$ степеня не вище d, що залежать суттєво від усіх змінних. Для кожної такої пари (A, φ) слід покласти $g(x) = \varphi(xA)$, $x \in V_n$ та перевірити умову $d(f,g) \le 2^{-d}(1-\varepsilon)$, за якої включити функцію g до списку, що формується. Сформований таким чином список буде містити усі функції, що належать множині (3.3), а також деякі функції з множини (3.2). Зазначену **модифікацію алгоритму 3.1** доцільно використовувати в тому випадку, коли потужність m множини (3.4) є не надто великою в порівнянні з числом k. Наступне твердження дозволяє оцінити трудомісткість цього алгоритму.

Алгоритм 3.1

Вхідні дані: вектор значень функції $f \in B_n$, $\varepsilon \in (0,1)$, $d \in \mathrm{N}$, $k \in \mathbf{N}$, де $d < k < n$.

Етап 1. Використовуючи алгоритм швидкого перетворення Адамара (див., наприклад, [86], с. 217), побудувати множину $S_f(\mu_0) = \{\alpha_1, \ldots, \alpha_m\}$ вигляду (3.4) та впорядкувати її елементи так, щоб $|\hat{f}(\alpha_1)| \ge \cdots \ge |\hat{f}(\alpha_m)|$.

Етап 2. Вибрати довільну систему $A_1, \ldots, A_l$ представників усіх класів еквівалентності на множині $S_f(\mu_0)^{\langle k \rangle}$. Для будь-якого $i \in \overline{1,l}$ та кожної функції $\varphi \in B_k$ такої, що $\deg\varphi \le d$, $I_\varphi = \{0\}$, покласти $g(x) = \varphi(xA_i)$, $x \in V_n$ та перевірити умову $d(f,g) \le 2^{-d}(1-\varepsilon)$. Якщо вона виконується, включити функцію g до списку, що формується.

Результат: множина (3.3), яка складається з усіх функцій у сформованому списку.

Рис. 3.1. Алгоритм побудови множини високоймовірних k-вимірних наближень степеня $d < k$ булевої функції від n змінних

Твердження 3.6. Трудомісткість модифікованого алгоритму 3.1 складає

$$T'_{\varepsilon}(n,k,d) = O\left(2^n nk \binom{m}{k} N_{k,d} \right) \tag{3.7}$$

операцій, де $m = |S_f(\mu_0)|$, $\mu_0 = \max\{2^{1-k}\varepsilon,\ \frac{4}{3\sqrt{3}} 2^{-k/2-d/2}\varepsilon^{3/2}\}$,

$N_{k,d} = \sum_{l=0}^{k} (-1)^l \binom{k}{l} 2^{\sum_{i=0}^{d}\binom{k-l}{i}}$ – кількість функцій $\varphi : V_k \to \{0,1\}$ степеня не вище d, які залежать суттєво від кожної змінної.

Доведення. На першому етапі для знаходження множини (3.4) та впорядкування її елементів достатньо виконати $T^{(1)} = O(2^n n)$ операцій (додавання, віднімання та порівняння дійсних чисел). На другому етапі треба перебрати $\binom{m}{k} N_{k,d}$ зазначених вище пар (A, φ), для кожної з яких перевірити умову лінійної незалежності стовпців матриці A та (за цієї умови) обчислити значення функції $g(x) = \varphi(xA)$, $x \in V_n$ і перевірити нерівність $d(f,g) \le 2^{-d}(1-\varepsilon)$. Остання процедура вимагає $O(2^n nk)$ операцій (арифметичних та булевих додавань, порівнянь дійсних чисел і звернень до функції φ). Отже, трудомісткість другого етапу алгоритму є $T^{(2)} = O\left(2^n nk \binom{m}{k} N_{k,d} \right)$. Звідси, враховуючи рівність $T'_{\varepsilon}(n,k,d) = T^{(1)} + T^{(2)}$, отримаємо формулу (3.7). Нарешті, вираз параметра $N_{k,d}$ отримується шляхом стандартного застосування методу включення-виключення (див., наприклад, [158], с. 65).

Твердження доведено.

У випадку, коли $d = k$ можна запропонувати більш ефективний алгоритм побудови множини (3.3). Зауважимо, що в цьому випадку зазначена множина складається з усіх строго k-вимірних функцій, які знаходяться від функції f на відносній відстані не більше ніж $2^{-k}(1-\varepsilon)$, а параметр μ_0 дорівнює $2^{1-k}\varepsilon$.

Для кожної матриці $A \in S_f(\mu_0)^{\langle k \rangle}$ позначимо φ_A^* булеву функцію, що визначається за правилом

$$\varphi_A^*(s) = 1 \;\Leftrightarrow \sum_{x \in V_n : xA = s} f(x) \geq 2^{n-k-1},\ s \in V_k \tag{3.8}$$

та покладемо $g_A^*(x) = \varphi_A^*(xA),\ x \in V_n$. Неважко переконатися в тому (див. доведення леми 3 в [143]), що для будь-якої функції $g(x) = \varphi(xA),\ x \in V_n$ виконується нерівність $d(f, g_A^*) \leq d(f, g)$. Крім того, на підставі наслідку 7 у [143] кожна така функція, що належить множині (3.3), відрізняється від функції g_A^* не більше ніж на одному вхідному наборі. Отже, для побудови множини (3.3) при $d = k$ можна використовувати алгоритм 3.2 (рис. 3.2).

Твердження 3.7. Трудомісткість алгоритму 3.2 складає

$$T_\varepsilon''(n,k) = O\left(2^{n+k} nk \binom{m}{k}\right) \tag{3.9}$$

операцій, де $m = |S_f(\mu_0)|,\ \mu_0 = 2^{1-k}\varepsilon$.

Доведення. Перший етап алгоритму 3.2 співпадає з аналогічним кроком алгоритму 3.1. Отже трудомісткість цього етапу складає $T^{(1)} = O(2^n n)$ операцій (додавання, віднімання та порівняння дійсних чисел). На другому етапі

потрібно перебрати $\binom{m}{k}$ матриць A, для кожної з яких перевірити умову лінійної незалежності її стовпців, що вимагає $O(nk^2)$ (двійкових) операцій.

Алгоритм 3.2

Вхідні дані: вектор значень функції $f \in B_n$; $\varepsilon \in (0,1)$, $k \in \mathbf{N}$, де $k < n$.

Етап 1. Покласти $\mu_0 = 2^{1-k}\varepsilon$; використовуючи алгоритм швидкого перетворення Адамара, побудувати множину $S_f(\mu_0) = \{\alpha_1, \ldots, \alpha_m\}$ вигляду (3.4) та впорядкувати її елементи так, щоб $|\hat{f}(\alpha_1)| \geq \cdots \geq |\hat{f}(\alpha_m)|$.

Етап 2. Для кожної матриці $A \in S_f(\mu_0)^{\langle k \rangle}$ обчислити значення функції $g_A^*(x) = \varphi_A^*(xA)$, $x \in V_n$ за формулою (3.8) та перевірити умову $d(f, g_A^*) \leq 2^{-k}(1-\varepsilon)$. Якщо вона виконується, то

– включити функцію g до списку, що формується;

– для кожного $s \in V_k$ обчислити значення функції $g_s(x) = \varphi_s(xA)$, $x \in V_n$, де $\varphi_s(s) = \varphi_A^*(s) \oplus 1$, $\varphi_s(s') = \varphi_A^*(s)$, якщо $s' \in V_k \setminus \{s\}$, та перевірити умову $d(f, g_s) \leq 2^{-k}(1-\varepsilon)$. Якщо ця умова виконується, включити функцію g_s до списку, що формується.

Результат: множина (3.3), яка складається з усіх функцій у сформованому списку.

Рис. 3.2. Алгоритм побудови множини високоймовірних k-вимірних наближень степеня $d = k$ булевої функції від n змінних

Далі, для кожної фіксованої матриці $A \in S_f(\mu_0)^{\langle k \rangle}$ слід перебрати не більше ніж $2^k + 1$ функцій вигляду g_A^*, g_s, де $s \in V_k$, та порівняти відносну

відстань між кожною з них і функцією f з числом $2^{-k}(1-\varepsilon)$. Остання процедура вимагає $O((2^k+1)2^n nk)$ операцій (арифметичних та булевих додавань, порівнянь дійсних чисел і звернень до функції f). Отже, трудомісткість другого етапу алгоритму є $T^{(2)} = O\left(2^{n+k} nk \binom{m}{k}\right)$. Звідси, враховуючи рівність $T''_\varepsilon(n,k) = T^{(1)} + T^{(2)}$, отримаємо формулу (3.9).

Твердження доведено.

На завершення цього пункту наведемо ще один допоміжний алгоритм, який дозволяє знайти функцію $g \in B_{n,d-1}$, розташовану від заданої функції $f \in B_n$ на відносній відстані не більше ніж $2^{-d}(1-\varepsilon)$. Відомо (див., наприклад, [23], п. 6), що існує не більше однієї такої функції, причому (за умови її існування) виконується рівність $I_g = \{\alpha \in V_n : \Delta_f(\alpha) \geq 1 - 2^{2-d}(1-\varepsilon)\}$, де

$$\Delta_f(\alpha) = 2^{-n} \sum_{x \in V_n} (-1)^{f(x \oplus \alpha) \oplus f(x)}, \ \alpha \in V_n \tag{3.10}$$

є автокореляційна функція булевої функції f. Шукану функцію g можна визначити за формулою

$$g(x) = \varphi^*_A(xA), \ x \in V_n, \tag{3.11}$$

де A – $n \times l$-матриця, стовпці якої утворюють базис векторного простору, дуального до підпростору I_g, $l \leq d-1$, а функція φ^*_A визначається за формулою (3.8) з заміною в ній k на l.

Алгоритм знаходження функції g наведено на рис. 3.3.

Твердження 3.8. Трудомісткість алгоритму 3.3 складає

$$T_{\varepsilon}'''(n,d) = O\left(2^n (n^2 + nd)\right) \tag{3.12}$$

операцій.

Алгоритм 3.3

Вхідні дані: вектор значень функції $f \in B_n$; $\varepsilon \in (0,1)$, $k \in \mathbf{N}$, де $k < n$.

Етап 1. Обчислити значення (3.10) за допомогою алгоритму швидкого перетворення Адамара. Знайти базис $\alpha_1, \ldots, \alpha_l$ векторного простору $\{\alpha \in V_n : \Delta_f(\alpha) \geq 1 - 2^{2-d}(1-\varepsilon)\}^{\perp}$ за допомогою алгоритму Гауса. Якщо $l > d-1$ або $|\{\alpha \in V_n : \Delta_f(\alpha) \geq 1 - 2^{2-d}(1-\varepsilon)\}| \neq 2^{n-l}$, закінчити роботу.

Етап 2. Сформувати з вектор-стовпців $\alpha_1, \ldots, \alpha_l$ матрицю A, визначити функцію g за формулою (3.11) та перевірити умову $d(f,g) \leq 2^{-d}(1-\varepsilon)$, за якої включити цю функцію до списку, що формується.

Результат: множина, що складається не більше ніж з однієї функції $g \in B_{n,d-1}$, яка задовольняє умові $d(f,g) \leq 2^{-d}(1-\varepsilon)$.

Рис. 3.3. Алгоритм знаходження $(d-1)$-вимірного високоймовірного наближення булевої функції від n змінних

Доведення. Для побудови множини $\{\alpha \in V_n : \Delta_f(\alpha) \geq 1 - 2^{2-d}(1-\varepsilon)\}$ на етапі 1 достатньо виконати $O(2^n n)$ операцій (додавання, віднімання та порівняння дійсних чисел), а для знаходження векторів $\alpha_1, \ldots, \alpha_l$ – ще $O(2^n n^2)$

(двійкових) операцій. Отже, трудомісткість етапу 1 складає $O(2^n n^2)$. Нарешті, оскільки трудомісткість етапу 2 дорівнює $O(2^n nl) = O(2^n nd)$, то трудомісткість всього алгоритму визначається за формулою (3.12).

Твердження доведено.

3.1.2. Метод розв'язання основної задачі та його порівняння з методом, що базується на алгоритмі Гопалана. Викладені вище результати дозволяють запропонувати такий метод побудови множини (3.2) за вектором значень функції f та числами d, k і ε. Помітимо, що ця множина є об'єднанням $k-d+2$ множин, що не перетинаються:

$$B_{n,k,d}(f;\varepsilon) = \{g \in B_{n,d-1} : d(f,g) \le 2^{-d}(1-\varepsilon)\} \cup \overline{B}_{n,d,d}(f;\varepsilon) \cup \left(\bigcup_{i=d+1}^{k} \overline{B}_{n,i,d}(f;\varepsilon) \right).$$

Отже, для побудови зазначених множин можна скористатися запропонованими вище алгоритмами (рис. 3.4).

Алгоритм 3.4

Вхідні дані: вектор значень функції $f \in B_n$; $\varepsilon \in (0,1)$, $k \in \mathrm{N}$, де $k < n$.

Етап 1. Для кожного $i \in \overline{d+1,k}$ застосувати модифікований алгоритм 3.1 до вхідних даних f, ε, d, i.

Етап 2. Застосувати алгоритм 3.2 до вхідних даних f, ε, d.

Етап 3. Застосувати алгоритм 3.3 до вхідних даних f, ε, d.

Етап 4. Об'єднати множини, отримані на етапах 1, 2, 3.

Результат: множина (3.2).

Рис. 3.4. Алгоритм реалізації запропонованого методу побудови множини (3.2)

Безпосередньо з тверджень 3.6 – 3.8 випливає такий результат.

Твердження 3.9. Трудомісткість алгоритму 3.4 складає

$$T_{n.k.d}^{(\varepsilon)} = O\left(2^n (n^2 + nd) + 2^{n+d} nd \binom{m_d}{d} + 2^n n \sum_{i=d+1}^{k} i \binom{m_i}{i} N_{i,d} \right) \qquad (3.13)$$

операцій, де

$$m_i = |S_f(\mu_{0,i})|,\ \mu_{0,i} = \max\{2^{1-i}\varepsilon,\ \frac{4}{3\sqrt{3}} 2^{-i/2-d/2} \varepsilon^{3/2}\},\ i \in \overline{d,k};$$

$$N_{i,d} = \sum_{l=0}^{i} (-1)^l \binom{i}{l} 2^{\sum_{j=0}^{d} \binom{i-l}{j}},\ i \in \overline{d+1,k}.$$

Порівняємо трудомісткість алгоритму 3.4 з трудомісткістю алгоритму, запропонованого П. Гопаланом [24]. Нагадаємо, що останній полягає у формуванні множини $S_f(\mu) = \{\alpha \in V_n : |\hat{f}(\alpha)| \ge \mu\}$, де $\mu = \frac{1}{8\sqrt{2}} 2^{-k/2-d} \varepsilon^2$, переборі усіх наборів $(\alpha_1, \dots, \alpha_k, \varphi)$, де $\alpha_1, \dots, \alpha_k \in S_f(\mu)$, $\varphi \in B_k$, $\deg\varphi \le d$, та перевірці умови $d(f,g) \le 2^{-d}(1-\varepsilon)$ для функції $g(x) = \varphi(\alpha_1 x, \dots, \alpha_k x)$, $x \in V_n$. Трудомісткість цього алгоритму складає не менше ніж

$$\tilde{T}_{n.k.d}^{(\varepsilon)} = 2^n m^k nk\ N(k,d) \qquad (3.14)$$

операцій (того ж самого типу, що використовуються в алгоритмах 3.1 – 3.3), де $m = |S_f(\mu)|$, $N(k,d) = 2^{\sum_{i=0}^{d} \binom{k}{i}}$ – число булевих функцій степеня не вище d від k змінних.

Як видно з формул (3.13), (3.14), трудомісткості обох алгоритмів суттєво залежать від спектру Уолша-Адамара вхідної функції f, а саме, від чисел $m_d, \ldots, m_k, m$. Для порівняння ефективності алгоритмів отримаємо більш прості (але більш грубі) оцінки їх трудомісткості. Помітимо, що в силу означень параметрів $\mu_{0,i}$ $(i \in \overline{d,k})$ та μ справедливі нерівності $\mu_{0,d} > \ldots > \mu_{0,k} > \mu$, з яких випливає, що $S_f(\mu_{0,d}) \subseteq \cdots \subseteq S_f(\mu_{0,d}) \subseteq S_f(\mu_{0,d})$ і, отже, $m_d \le \ldots \le m_k \le m$. Таким чином, на підставі формул (3.13), (3.14) трудомісткість алгоритму 3.4 не перевищує значення

$$T^{(\varepsilon)}_{n.k.d}(m_k) = O\left(2^n(n^2+nd) + 2^{n+d}\,nd\binom{m_k}{d} + 2^n n\sum_{i=d+1}^{k} i\binom{m_k}{i}N_{i,d}\right), \qquad (3.15)$$

в той час як трудомісткість алгоритму Гопалана є не менше ніж

$$\widetilde{T}^{(\varepsilon)}_{n.k.d} = 2^n (m_k)^k\, nk\; N(k,d), \qquad (3.16)$$

де $m_k = |\{\alpha \in V_n : |\hat{f}(\alpha)| \ge \mu_{0,k}\}|$, $\mu_{0,k} = \max\{2^{1-k}\varepsilon,\ \frac{4}{3\sqrt{3}}2^{-k/2-d/2}\varepsilon^{3/2}\}$.

Зазначимо, що обидва вирази (3.15), (3.16) залежать від єдиного параметра m_k, який визначається вхідною функцією f та може приймати цілочисельні значення від 0 до $(\mu_{0,k})^{-2}$.

В табл. 3.1 наведено значення параметрів (3.15) та (3.16), отримані для низки значень n, k, d, m_k і ε.

Як видно з таблиці, алгоритм 3.4 дозволяє будувати k-вимірні наближення булевих функцій більш ефективно в порівнянні з алгоритмом Гопалана (у певних випадках – в 1000 та більше разів) і може бути застосований на практиці при малих значеннях k і d, якщо кількість m_k "відносно великих" за модулем

коефіцієнтів Уолша-Адамара функції f не перевищує 20. У випадку $d = k$ трудомісткість алгоритму 3.4 помітно зменшується, що робить можливим його застосування при більших значеннях k (наприклад, $k = 10$), в той час як алгоритм Гопалана стає практично незастосовним.

Таблиця 3.1

Результати порівняння ефективностей алгоритму 3.4 та алгоритму Гопалана ($\varepsilon = 0{,}125$)

Параметри					$T^{(\varepsilon)}_{n.k.d}(m_k)$	$\widetilde{T}^{(\varepsilon)}_{n.k.d}$
n	k	d	$\log(\mu_{0,k})^{-2}$	m_k		
10	4	2	12,00	5	2^{29}	2^{36}
				20	2^{39}	2^{44}
		3	12,00	5	2^{33}	2^{40}
				20	2^{46}	2^{48}
		4	12,00	5	2^{22}	2^{41}
				20	2^{32}	2^{49}
	5	2	18,00	6	2^{35}	2^{45}
				20	2^{46}	2^{54}
		4	18,00	6	2^{50}	2^{60}
				20	2^{61}	2^{69}
		5	18,00	6	2^{24}	2^{61}
				20	2^{35}	2^{70}
15	4	2	12,00	5	2^{35}	2^{42}
				20	2^{44}	2^{50}
		3	12,00	5	2^{39}	2^{46}
				20	2^{49}	2^{54}
		4	12,00	5	2^{28}	2^{47}
				20	2^{38}	2^{55}
	5	2	18,00	6	2^{40}	2^{51}
				20	2^{51}	2^{59}
		4	18,00	6	2^{55}	2^{66}
				20	2^{67}	2^{74}
		5	18,00	6	2^{29}	2^{67}
				20	2^{41}	2^{75}

3.1.3. Приклад застосування запропонованого методу. Алгоритм 3.4 було реалізовано програмно та застосовано до аналізу кореляційних властивостей низки булевих функцій від невеликої кількості змінних. Як приклад, що ілюструє отримані результати, розглянемо функцію f від п'яти змінних з класу Карле-Фенга [159] (табл. 3.2). Вибір цієї функції обумовлено її гарними криптографічними властивостями, зокрема, високою нелінійністю (тобто відстанню до класу $B_{n,1}$).

Таблиця 3.2

Функція Карле-Фенга від $n = 5$ змінних

x	$f(x)$	x	$f(x)$	x	$f(x)$	x	$f(x)$
00000	1	01000	1	10000	1	11000	0
00001	1	01001	1	10001	0	11001	0
00010	1	01010	0	10010	1	11010	0
00011	1	01011	1	10011	1	11011	0
00100	1	01100	1	10100	0	11100	0
00101	0	01101	0	10101	0	11101	0
00110	1	01110	0	10110	1	11110	1
00111	0	01111	1	10111	0	11111	0

В табл. 3.3 показано ненормовані коефіцієнти Уолша-Адамара даної функції, впорядковані за убуванням модулів їх значень. Зрозуміло, що при $d = k$ та $0 < \varepsilon \le 1/4$ множина $S_f(\mu_0)$ вигляду (3.4) складається з усіх векторів α в табл. 3.3, які задовольняють умові $\hat{f}(\alpha) > 0$. В цьому випадку застосування алгоритму 3.4 для кожного $k = 2, 3, 4$ показує відсутність k-вимірних наближень функції f, які знаходяться від неї на відносній відстані не більше ніж $2^{-k}(1-\varepsilon)$.

Таблиця 3.3

Коефіцієнти Уолша-Адамара функції f

α	$32\hat{f}(\alpha)$	α	$32\hat{f}(\alpha)$	α	$32\hat{f}(\alpha)$	α	$32\hat{f}(\alpha)$
10000	– 12	01100	– 8	10101	4	10100	– 4
00101	8	10010	– 8	11000	4	11001	– 4
10111	8	10011	– 8	11100	4	00000	0
11011	8	00010	4	11101	4	01101	0
00001	– 8	01010	4	00011	– 4	10110	0
00100	– 8	01011	4	00110	– 4	11010	0
01000	– 8	01110	4	00111	– 4	11110	0
01001	– 8	10001	4	01111	– 4	11111	0

Таблиця 3.4

Результати дослідження k-вимірних наближень функції f

k	Відстань до найближчої функції (3.1) такої, що $A \in S_f(\mu_0)^{\langle k \rangle}$	Кількість наближень на цій відстані	Приклади наближень: φ, A^T
2	10	26	$1 \oplus x_1 \oplus x_2$, $\begin{pmatrix} 10011 \\ 00011 \end{pmatrix}$
3	8	68	$1 \oplus x_2 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1x_2x_3$, $\begin{pmatrix} 00011 \\ 00111 \\ 10100 \end{pmatrix}$
4	6	84	$1 \oplus x_2 \oplus x_3 \oplus x_1x_3 \oplus x_1x_4 \oplus x_2x_4 \oplus x_1x_2x_3 \oplus \oplus x_1x_2x_4 \oplus x_1x_3x_4 \oplus x_2x_3x_4 \oplus x_1x_2x_3x_4$, $\begin{pmatrix} 01110 \\ 10101 \\ 11100 \\ 11001 \end{pmatrix}$

Поряд з тим, запропонований метод дозволяє отримати інформацію про найкращі наближення (3.1) функції f, які задаються матрицями $A \in S_f(\mu_0)^{\langle k \rangle}$ (див. позначення перед формулюванням твердження 3.5) та довільними функціями $\varphi \in B_k$. Результати представлено в табл. 3.4 (обчислення проведені на ПК з процесором Intel (R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам`яті 4 ГБ RAM на базі 32-розрядної ОС Windows 7 Service Pack 1 в середовищі Microsoft Visual Studio 2010 (.NET Framefork 4.0, мова програмування – C#); час виконання алгоритму для кожного k складає декілька секунд).

Як видно з табл. 3.4, при $k = 2$ існує 26 k-вимірних функцій зазначеного вище вигляду, кожна з яких знаходиться від функції f на найближчій відстані, що дорівнює 10. З ростом k кількість зазначених наближень функції f помітно зростає, в той час як відстань до них зменшується.

Таким чином, запропонований метод може бути застосований на практиці при аналізі кореляційних властивостей функцій ускладнення синхронних потокових шифрів при малих значеннях k і d, якщо кількість "відносно великих" за модулем коефіцієнтів Уолша-Адамара функції f не перевищує 20.

3.2. Метод обґрунтування відсутності високоймовірних k-вимірних наближень булевих функцій

В попередньому підрозділі викладено метод побудови списку всіх k-вимірних функцій степеня не вище d, які знаходяться на відносній відстані не більше $2^{-d}(1-\varepsilon)$ від заданої булевої функції n змінних, $1 \le d \le k < n$, $\varepsilon \in (0,1)$. Показано, що зазначений метод є більш ефективним у порівнянні з методом, що базується на алгоритмі Гопалана [24] (у певних випадках – в 1000 та більше разів).

Поряд з тим, при обґрунтуванні стійкості синхронних потокових шифрів відносно статистичних атак, викладених в розділі 2, не обов'язково шукати k-вимірні наближення заданої булевої функції. Достатньо лише обґрунтовувати відсутність таких наближень, що знаходяться від заданої функції на відносно невеликій відстані. Ця задача зводиться до розробки методів оцінювання відносної відстані між заданою функцією та множиною всіх k-вимірних функцій від n змінних. Як зазначено в розділі 1, остання задача є нетривіальною навіть при $k = 2$, якщо n є достатньо великим числом (наприклад, $n \geq 64$). Зокрема, на сьогодні не відомо (детермінованих) алгоритмів її розв'язання, складність яких поліноміально залежить від n.

Основним науковим результатом даного підрозділу є метод обчислення значень нижніх меж відносної відстані між зрівноваженою булевою функцією та множиною всіх k-вимірних функцій від n змінних. Зазначений метод запропоновано вперше. Його сутність полягає у статистичному оцінюванні відносної відстані за допомогою спеціально розробленого ймовірнісного алгоритму, складність якого залежить лінійно від n та поліноміально від величин, обернених до точності та імовірності помилки алгоритму. Наведено результати моделювання цього алгоритму, які дозволяють стверджувати про можливість його застосування на практиці при обґрунтуванні стійкості функцій ускладнення синхронних потокових шифрів відносно статистичних атак, наведених в розділі 2 .

Позначимо $V_n^* = V_n \setminus \{0\}$. На підставі теореми 1 в [143] відносна відстань між функцією $f \in B_n$ та множиною $B_{n,k}$ визначається за формулою

$$d(f, B_{n,k}) = 1/2 \cdot \left(1 - \max_{H \in L_{n,k}} l_f(H)\right), \qquad (3.17)$$

де максимум береться за всіма k-вимірними підпросторами H векторного простору V_n,

$$l_f(H) = 2^{-k} \sum_{s \in V_k} \left| \sum_{x \in H} \hat{f}(x)(-1)^{\alpha_s x} \right|, \tag{3.18}$$

а $\{\alpha_s : s \in V_k\}$ є система представників усіх суміжних класів простору V_n по підпростору $H^{\perp}$, дуальному до H.

Припустимо, що f є зрівноваженою функцією, тобто задовольняє умові $\hat{f}(0) = 0$. Позначимо H^* довільний підпростір, на якому досягається максимум значень (3.18) у правій частині рівності (3.17) та зафіксуємо $n \times k$-матрицю A, стовпці якої утворюють базис підпростору H^*. Тоді рівність (3.17) можна записати у вигляді

$$d(f, B_{n,k}) = 1/2 \cdot (1 - l_{f,A}), \tag{3.19}$$

де

$$l_{f,A} = 2^{-k} \sum_{s \in V_k} \left| \sum_{y \in V_k^*} \hat{f}(Ay)(-1)^{sy} \right|. \tag{3.20}$$

Припустимо зараз, що (зрівноважена) функція f задається за допомогою оракула, тобто певного алгоритму, що дозволяє обчислювати значення $f(x)$ для довільного вхідного набору $x \in V_n$. Треба розробити алгоритм обчислення для будь-яких $k \in \overline{1, n-1}$, $\varepsilon, \delta \in (0,1)$ статистичної верхньої оцінки параметра (3.20) з точністю ε та надійністю не менше $1 - \delta$, тобто такого випадкового значення $\theta \in (0,1)$, що задовольняє умові

$$\mathbf{P}\{l_{f,A} \le \theta + \varepsilon\} \ge 1 - \delta. \tag{3.21}$$

При цьому вимагається, щоб для будь-якого фіксованого k обчислювальна складність алгоритму поліноміально залежала від n, ε^{-1} та δ^{-1}.

Для викладення алгоритму, що пропонується, введемо низку позначень та доведемо одну теорему.

Зафіксуємо натуральне число $t > k$ та позначимо X випадкову матрицю з рівномірним розподілом ймовірностей на множині $F_{t\times n}$. Розглянемо випадкові величини

$$\xi_a(X) = \frac{1}{2^t - 1}\sum_{u\in V_t^*}(-1)^{f(uX)\oplus ua},\ a \in V_t, \tag{3.22}$$

$$\eta_B(X) = 2^{-k}\sum_{s\in V_k}\left|\sum_{y\in V_k^*}\xi_{By}(X)(-1)^{sy}\right|,\ B \in F_{t\times k}. \tag{3.23}$$

Назвемо матрицю $B \in F_{t\times k}$ спеціальною ступеневою, якщо транспонована до неї матриця має вигляд

$$B^T = \begin{pmatrix} & \overset{i_1}{} & & & \overset{i_2}{} & & & \overset{i_r}{} & \\ 0\ldots0 & 1 & * & \ldots & * & 0 & * & \ldots & * \\ 0\ldots0 & 0 & 0\ldots0 & & 1 & * & \ldots & * & 0 & * & \ldots & * \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0\ldots0 & 0 & 0\ldots0 & & 0 & 0\ldots0 & & 1 & * & \ldots & * \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0\ldots0 & 0 & 0\ldots0 & & 0 & 0\ldots0 & 0 & 0\ldots0 & & & \end{pmatrix},$$

де $*$ позначає довільний елемент поля F, $1 \le i_1 < \ldots < i_r \le t$. Добре відомо (див., наприклад, [160], с. 149), що будь-яка матриця $B \in F_{t\times k}$ є стовпцево еквівалентною єдиній спеціальній ступеневій матриці, тобто існує точно одна

зазначена матриця $\widetilde{B}$, що пов'язана з B співвідношенням $\widetilde{B} = BU$, де U – оборотна матриця порядку k над полем F.

Позначимо $\widetilde{F}_{t\times k}$ множину усіх спеціальних ступеневих матриць розміру $t\times k$ над F та покладемо

$$\theta(X) = \max_{B\in\widetilde{F}_{t\times k}} \{\eta_B(X)\} \tag{3.24}$$

Теорема 3.1. Нехай $k\in\overline{1,n-1}$, $\varepsilon,\delta\in(0,1)$ і $2^t \geq 2^k\varepsilon^{-2}\delta^{-1}$. Тоді для випадкової величини $\theta = \theta(X)$ виконується нерівність (3.21).

Доведення. Перш за все, переконаємося у справедливості такої нерівності:

$$|l_{f,A} - \eta_{XA}(X)| \leq \left(\sum_{y\in V_k^*} (\hat{f}(Ay) - \xi_{XAy}(X))^2\right)^{1/2}. \tag{3.25}$$

Дійсно, на підставі формул (3.20), (3.22) та нерівності між середнім арифметичним і середнім квадратичним

$$\left|l_{f,A} - \eta_{XA}(X)\right| = \left|2^{-k}\sum_{s\in V_k}\left(\left|\sum_{y\in V_k^*}\hat{f}(Ay)(-1)^{sy}\right| - \left|\sum_{y\in V_k^*}\xi_{XAy}(X)(-1)^{sy}\right|\right)\right| \leq$$

$$\leq 2^{-k}\sum_{s\in V_k}\left|\sum_{y\in V_k^*}\left(\hat{f}(Ay) - \xi_{XAy}(X)\right)(-1)^{sy}\right| \leq$$

$$\leq \left(2^{-k}\sum_{s\in V_k}\left(\sum_{y\in V_k^*}\left(\hat{f}(Ay) - \xi_{XAy}(X)\right)(-1)^{sy}\right)^2\right)^{1/2}. \tag{3.26}$$

Далі,

$$2^{-k}\sum_{s\in V_k}\left(\sum_{y\in V_k^*}\left(\hat f(Ay)-\xi_{XAy}(X)\right)(-1)^{sy}\right)^2=$$

$$=2^{-k}\sum_{y_1,y_2\in V_k^*}\left(\hat f(Ay_1)-\xi_{XAy_1}(X)\right)\left(\hat f(Ay_2)-\xi_{XAy_2}(X)\right)\times$$

$$\times\sum_{s\in V_k}(-1)^{s(y_1\oplus y_2)}=\sum_{y\in V_k^*}\left(\hat f(Ay)-\xi_{XAy}(X)\right)^2.$$

Підставляючи зазначений вираз у формулу (3.26), отримаємо нерівність (3.25).

Покажемо зараз, що

$$\mathrm{P}_X\{|\,l_{f,A}-\eta_{XA}(X)\,|>\varepsilon\}\le 2^{k-t}\varepsilon^{-2}. \tag{3.27}$$

Дійсно, на підставі формули (3.25) та нерівності Маркова справедливі такі оцінки:

$$\mathbf{P}_X\{|\,l_{f,A}-\eta_{XA}(X)\,|>\varepsilon\}\le\mathbf{P}_X\left\{\sum_{y\in V_k^*}(\hat f(Ay)-\xi_{XAy}(X))^2>\varepsilon^2\right\}\le$$

$$\le\varepsilon^{-2}\sum_{y\in V_k^*}\mathbf{E}(\hat f(Ay)-\xi_{XAy}(X))^2. \tag{3.28}$$

Далі, безпосередньо з формули (3.22) випливає, що для будь-якого $y\in V_k^*$ математичне сподівання випадкової величини $\xi_{XAy}(X)$ дорівнює $\hat f(Ay)$. Отже,

$$\sum_{y\in V_k^*}\mathbf{E}(\hat{f}(Ay)-\xi_{XAy}(X))^2=\sum_{y\in V_k^*}\mathbf{E}(\xi_{XAy}(X)-\mathbf{E}(\xi_{XAy}(X))^2=$$

$$=\sum_{y\in V_k^*}\mathbf{D}(\xi_{XAy}(X))=\frac{1}{(2^t-1)^2}\sum_{y\in V_k^*}\mathbf{D}\left(\sum_{u\in V_t^*}(-1)^{f(uX)\oplus uX(Ay)}\right)=$$

$$=\frac{1}{(2^t-1)^2}\sum_{y\in V_k^*}\sum_{u\in V_t^*}\mathrm{D}\left((-1)^{f(uX)\oplus uX(Ay)}\right),$$

де остання рівність випливає з того, що доданки в сумі під знаком дисперсії є попарно незалежними випадковими величинами. Нарешті, використовуючи тривіальну оцінку $\mathrm{D}\left((-1)^{f(uX)\oplus uX(Ay)}\right)\le 1$, $u\in V_t^*$, $y\in V_k^*$, отримаємо, що

$$\sum_{y\in V_k^*}\mathbf{E}(\hat{f}(Ay)-\xi_{XAy}(X))^2\le\frac{2^k-1}{2^t-1}\le 2^{k-t}.$$

Звідси на підставі формули (3.28) випливає нерівність (3.27).

Для завершення доведення залишається зауважити, що $\eta_B(X)=\eta_{BU}(X)$ для будь-яких матриці $B\in F_{t\times k}$ та оберненої матриці $U\in F_{k\times k}$. Отже, згідно з формулою (3.24), $\theta(X)\ge\eta_B(X)$ для довільної (а не тільки спеціальної ступеневої) матриці $B\in F_{t\times k}$. Зокрема, подія $\{l_{f,A}>\theta(X)+\varepsilon\}$ тягне подію $\{l_{f,A}>\eta_{XA}(X)+\varepsilon\}$, звідки на підставі нерівності (3.27) випливає, що

$$\mathbf{P}_X\{l_{f,A}>\theta(X)+\varepsilon\}\le\mathbf{P}_X\{l_{f,A}>\eta_{XA}(X)+\varepsilon\}\le\mathbf{P}_X\{|\,l_{f,A}-\eta_{XA}(X)\,|>\varepsilon\}\le 2^{k-t}\varepsilon^{-2}$$

Таким чином, за умови $2^t\ge 2^k\varepsilon^{-2}\delta^{-1}$ справедлива нерівність (3.21). Теорему доведено.

Наведемо зараз ймовірнісний алгоритм обчислення нижніх меж параметра (3.19).

Алгоритм 3.5

Вхідні дані: зрівноважена функція $f \in B_n$, задана за допомогою оракула; $k \in \overline{1, n-1}$, $\varepsilon, \delta \in (0,1)$.

Етап 1. Покласти

$$t = k + \lceil \log(\varepsilon^{-2}\delta^{-1}) \rceil, \tag{3.29}$$

згенерувати випадкову рівноймовірну булеву $t \times n$-матрицю X та обчислити значення функції $f_X(u) = f(uX)$, $u \in V_t$.

Етап 2. Обчислити значення (3.22) за допомогою алгоритму швидкого перетворення Адамара.

Етап 3. Для кожної матриці $B \in \widetilde{F}_{t \times k}$:

– за відомими значеннями (3.22) обчислити значення функції

$$h(y) = \begin{cases} \xi_{By}(X), \text{ якщо } y \in V_k^*; \\ 0, \text{ якщо } y = 0; \end{cases}$$

– обчислити значення $\hat{h}(s) = \sum_{y \in V_k^*} \xi_{By}(X)(-1)^{sy}$, $s \in V_k$ за допомогою алгоритму швидкого перетворення Адамара;

– покласти $\eta_B(X) = 2^{-k} \sum_{s \in V_k} |\hat{h}(s)|$.

Етап 4. Покласти $\theta(X) = \max\limits_{B \in \widetilde{F}_{t \times k}} \{\eta_B(X)\}$.

Результат: випадкове число $\Delta(f, B_{n,k}) = 1/2 \cdot (1 - (\theta(X) + \varepsilon))$, що задовольняє умові $\mathbf{P}_X\{d(f, B_{n,k}) \geq \Delta(f, B_{n,k})\} \geq 1 - \delta$.

Рис. 3.5. Алгоритм обчислення нижніх меж відносної відстані між зрівноваженою булевою функцією та множиною k-вимірних функцій

Коректність алгоритму 3.5 випливає з наведеної вище теореми.

Для оцінювання трудомісткості цього алгоритму доведемо допоміжне твердження.

Лема 3.1. Для будь-яких натуральних $k < t$ справедлива нерівність

$$| \widetilde{F}_{t\times k} | \le c(k+1)\max\{2^{k^2}, 2^{k(t-k)}\},$$

де $c^{-1} = \prod_{i=1}^{\infty}(1-2^{-i})$.

Доведення. З означення множини $\widetilde{F}_{t\times k}$ випливає, що $| \widetilde{F}_{t\times k} | = \sum_{i=0}^{k} \begin{bmatrix} t \\ i \end{bmatrix}$, де

$$\begin{bmatrix} t \\ i \end{bmatrix} = \frac{(2^t-1)(2^{t-1}-1)\cdots(2^{t-i+1}-1)}{(2^i-1)(2^{i-1}-1)\cdots(2-1)}$$

є число i-вимірних підпросторів простору V_t, $i \in \overline{0,t}$. Використовуючи оцінку $\begin{bmatrix} t \\ i \end{bmatrix} \le c \cdot 2^{i(t-i)}$, $i \in \overline{0,t}$, отримаємо, що

$$| \widetilde{F}_{t\times k} | = \sum_{i=0}^{k} \begin{bmatrix} t \\ i \end{bmatrix} \le c\sum_{i=0}^{k} 2^{i(t-i)} \le c(k+1)2^{t^2/4}.$$

Нехай $t \le 2k$. Тоді в силу наведеної нерівності $| \widetilde{F}_{t\times k} | \le c(k+1)2^{k^2}$. Якщо ж $t > 2k$, то

$$\begin{bmatrix} t \\ i \end{bmatrix} \le \begin{bmatrix} t \\ k \end{bmatrix} \le c \cdot 2^{k(t-k)}$$

для кожного $i \in \overline{0,k}$ і, отже, $|\widetilde{F}_{t\times k}| \le c(k+1)2^{k(t-k)}$. Таким чином, в будь-якому випадку виконується нерівність

$$|\widetilde{F}_{t\times k}| \le c(k+1)\max\{2^{k^2}, 2^{k(t-k)}\},$$

що й треба було довести.

Теорема 3.2. Трудомісткість алгоритму 3.5 в найгіршому випадку дорівнює

$$T_{n,k}(\varepsilon,\delta) = O\left(2^k(k+\varepsilon^{-2}\delta^{-1})(n\varepsilon^{-2}\delta^{-1}+k^2\max\{2^{k^2}, \varepsilon^{-2k}\delta^{-k}\})\right) \qquad (3.30)$$

Доведення. На етапі 1 потрібно виконати $O(2^t tn)$ операцій (булевого додавання та звернення до функції f), а на етапі 2 – $O(2^t t)$ операцій (додавання та віднімання цілих чисел). Далі, для кожної фіксованої матриці $B \in \widetilde{F}_{t\times k}$ на етапі 3 потрібно виконати $O(2^k tk)$ таких самих операцій. Отже, на підставі леми 3.1 трудомісткість етапу 3 складає

$$O(2^k tk|\widetilde{F}_{t\times k}|) = O(2^k tk^2\max\{2^{k^2}, 2^{k(t-k)}\}).$$

Нарешті, трудомісткість четвертого етапу є $O(|\widetilde{F}_{t\times k}|) = O(\max\{2^{k^2}, 2^{k(t-k)}\})$. Підсумовуючи отримані оцінки, з урахуванням рівності (3.29) отримаємо формулу (3.30).

Теорему доведено.

В табл. 3.5 показано чисельні значення двійкового логарифму виразу під знаком O в правій частині рівності (3.30). Як видно з таблиці, з ростом параметра k чи зменшенням параметра ε трудомісткість алгоритму досить

швидко зростає. Навпаки, помітне збільшення параметра n практично не впливає на зростання трудомісткості алгоритму.

У табл. 3.6, 3.7 показано результати застосування алгоритму 3.5 при $k = 2$ до функцій від 64 та 11 змінних відповідно. Обчислення проводилися на ПК з процесором Intel Core i7 (1,6 ГГц) та обсягом оперативної пам'яті 4 Гб RAM (DDR3) на базі Windows 7 (використовувався пакет прикладних програм Maple 13.0). Середній час роботи комп'ютерної програми складає 804,5 секунди при $n = 64$ та 652,1 секунди при $n = 11$.

Таблиця 3.5

Чисельні оцінки трудомісткості запропонованого алгоритму $(\delta = 0{,}0625)$

k / ε	2		3		4	
$1/2$	20,3663	21,2143	22,7665	23,1536	30,0931	30,0987
$1/4$	25,0112	25,5962	30,2263	30,2647	36,0238	36,0252
$1/8$	30,3247	30,5878	38,1766	38,1791	46,0056	46,0057
n	64	128	64	128	64	128

У табл. 3.6 наведені значення

$$\Delta(f, B_{n,2}) = 1/2 \cdot (1 - (\theta(X) + \varepsilon))$$

статистичних нижніх оцінок параметра $d(f, B_{n,2})$, отримані за допомогою алгоритму 3.5 для двох функцій вигляду

$$f(x) = f(x_1, x_2) = g(x_1, x_2)h_1(x_2) \oplus (g(x_1, x_2) \oplus 1)h_2(x_2),\ x = (x_1, x_2) \in V_{n_1} \times V_{n_2},$$

де $g(x_1, x_2) = (\alpha_1 x_1)(\alpha_2 x_1)$ – строго 2-вимірна функція, що не залежить суттєво від останніх n_2 змінних (α_1 та α_2 є різними ненульовими булевими векторами довжини n_1, а $\alpha_i x_1$ позначає булев скалярний добуток векторів α_i та $x_1 \in V_{n_1}$, $i = 1, 2$); h_1 та h_2 – довільні булеві функції від n_2 змінних, що задовольняють умові $wt(h_1) = 2^{-n_2}(2^{n_2+1} - 3m)$, $wt(h_2) = 2^{-n_2} m$, де m – ціле число таке, що $2^{n_2} < 3m < 2^{n_2+1}$.

Таблиця 3.6

Статистичні нижні оцінки значення відстані $d(f, B_{n,2})$
($n_1 = 20$, $n_2 = 44$, $\varepsilon = 1/4$, $\delta = 0{,}0625$, $t = 10$)

Номер експерименту	m	$\theta(X)$	$\Delta(f, B_{n,2})$	$\Delta^*(f, B_{n,2})$
1	$(2^{44}+5)/3$	0,5010	0,1245	0,25
2		0,5010	0,1245	
3		0,5191	0,1155	
4		0,5308	0,1096	
5		0,5020	0,1240	
6	$(2^{45}-5)/3$	0,5020	0,1240	0,25
7		0,5059	0,1221	
8		0,5010	0,1245	
9		0,5020	0,1240	
10		0,5073	0,1213	

Як показує прямий підрахунок, в цьому випадку

$$wt(f) = 1/4 \cdot wt(h_1) + 3/4 \cdot wt(h_2) = 1/2,$$

$$d(f, g) = 1/4 \cdot (1 - wt(h_1)) + 3/4 \cdot wt(h_2) = 1/4 + 1/2 \cdot (1 - wt(h_1)).$$

Отже, f є зрівноваженою функцією від $n = n_1 + n_2$ змінних, що знаходиться від множини $B_{n,2}$ на відносній відстані не більше ніж

$$\Delta^*(f, B_{n,2}) = \min\{1/4 + 1/2 \cdot (1 - wt(h_1)),\ 3/4 - 1/2 \cdot (1 - wt(h_1))\}. \qquad (3.31)$$

Параметр m у табл. 3.6 задає функції h_1 та h_2 за правилом: $h_1(x_2) = 1 \Leftrightarrow |x_2| < 2^{n_2+1} - 3m$, $h_2(x_2) = 0 \Leftrightarrow |x_2| < 2^{n_2} - m$, де $|x_2|$ – двійкове ціле число, що відповідає вектору $x_2 \in V_{n_2}$. Значення t і $\theta(X)$ обчислені, відповідно, на етапах 1 і 4 алгоритму, а в останній колонці таблиці показані значення параметра (3.31).

Розглянемо функцію ускладнення, яка використовується в алгоритмі шифрування Achterbahn-80 [161] (рис. 3.6).

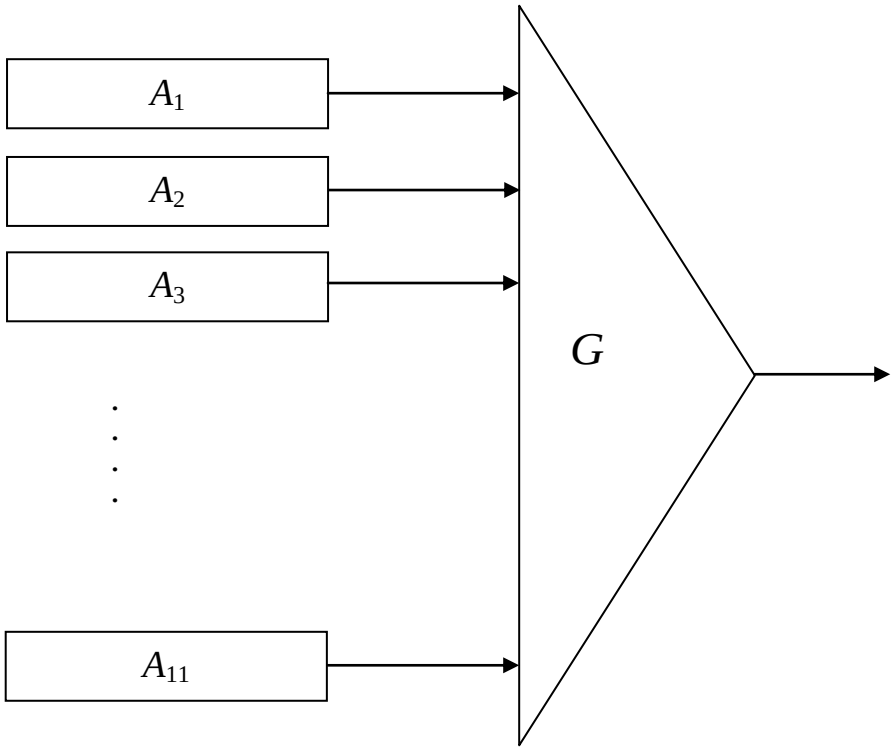

Рис. 3.6. Схема генератора гами алгоритму Achterbahn-80

Генератор гами цього шифру складається з 11 нелінійних регістрів зсуву $A_1, A_2, ..., A_{11}$ довжини 22, 23, …, 32 відповідно, та функції ускладнення вигляду:

$$G(x_1,..., x_{11}) = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_9 \oplus x_{11} \oplus x_2x_{10} \oplus x_2x_{11} \oplus x_4x_8 \oplus x_5x_6 \oplus$$

$$\oplus x_6x_8 \oplus x_6x_{10} \oplus x_6x_{11} \oplus x_7x_8 \oplus x_8x_9 \oplus x_8x_{10} \oplus x_9x_{10} \oplus x_9x_{11} \oplus x_1x_2x_8 \oplus x_1x_4x_{10} \oplus$$

$$\oplus x_1x_4x_{11} \oplus x_1x_8x_9 \oplus x_1x_9x_{10} \oplus x_1x_9x_{11} \oplus x_2x_3x_8 \oplus x_2x_4x_8 \oplus x_2x_4x_{10} \oplus x_2x_4x_{11} \oplus$$

$$\oplus x_2x_7x_8 \oplus x_2x_8x_{10} \oplus x_2x_8x_{11} \oplus x_2x_9x_{10} \oplus x_2x_9x_{11} \oplus x_3x_4x_8 \oplus$$

$$\oplus x_3x_8x_9 \oplus x_4x_7x_8 \oplus x_4x_8x_9 \oplus x_5x_6x_8 \oplus x_5x_6x_{10} \oplus x_5x_6x_{11} \oplus x_6x_8x_{10} \oplus x_6x_8x_{11}$$

$$\oplus x_7x_8x_9 \oplus x_8x_9x_{10} \oplus \oplus x_8x_9x_{11} \oplus x_1x_2x_3x_8 \oplus x_1x_2x_7x_8 \oplus x_1x_3x_5x_8 \oplus x_1x_3x_8x_9 \oplus$$

$$\oplus x_1x_4x_8x_{10} \oplus x_1x_4x_8x_{11} \oplus x_1x_5x_7x_8 \oplus x_1x_7x_8x_9 \oplus x_1x_8x_9x_{10} \oplus x_1x_8x_9x_{11} \oplus x_2x_3x_4x_8 \oplus$$

$$\oplus x_2x_3x_5x_8 \oplus x_2x_4x_7x_8 \oplus x_2x_4x_8x_{10} \oplus x_2x_4x_8x_{11} \oplus x_2x_5x_7x_8 \oplus x_2x_8x_9x_{10} \oplus x_2x_8x_9x_{11} \oplus$$

$$\oplus x_3x_4x_8x_9 \oplus x_4x_7x_8x_9 \oplus x_5x_6x_8x_{10} \oplus x_5x_6x_8x_{11}, (x_1,..., x_{11}) \in V_{11}.$$

Функція G є зрівноваженою, має степінь нелінійності 4, порядок кореляційної імунності 6, нелінійність 896 та максимальну алгебраїчну імунність, яка дорівнює 4. При цьому кожна змінна $x_1,..., x_{11}$ входить, принаймні, в один доданок степеня 4.

Табл. 3.7 демонструє результати застосування алгоритму 3.5 до функції G. Обчислення відносної відстані $d(G, B_{n,2})$ за формулою (3.17) показує, що її точне значення дорівнює 0,4375. При цьому час обчислення складає 3020 секунд, що приблизно в 5 разів перевищує середній час оцінювання цього параметра з точністю $\varepsilon = 1/8$ та надійністю $1 - \delta \geq 0{,}75$ (див. табл. 3.7).

Таблиця 3.7

Статистичні нижні оцінки параметра $d(G, B_{n,2})$
($n=11,\ \varepsilon=1/8,\ \delta=0{,}25,\ t=10$)

Номер експерименту	$\theta(X)$	$\Delta(G, B_{n,2})$	Номер експерименту	$\theta(X)$	$\Delta(G, B_{n,2})$
1	0,2517	0,3316	6	0,2512	0,3119
2	0,1881	0,3434	7	0,2498	0,3126
3	0,1881	0,3434	8	0,1891	0,3429
4	0,1266	0,3742	9	0,1891	0,3429
5	0,1881	0,3434	10	0,2507	0,3121

Для підвищення точності оцінок, наведених у табл. 3.6, слід зменшити значення ε, що призведе до збільшення часу виконання алгоритму. Наприклад, при $\varepsilon=1/8$ нижня оцінка параметра $d(f, B_{n,2})$ становить приблизно 0,1871, при цьому середній час роботи комп'ютерної програми складає майже 4 години.

В цілому, отримані результати показують, що при малих значеннях k запропонований метод може бути ефективно використаний на практиці для обґрунтуванні стійкості функцій ускладнення потокових шифрів відносно статистичних атак, наведених в розділі 2.

Висновки

1. Першим науковим результатом розділу є метод побудови списку всіх k-вимірних функцій степеня не вище d, які знаходяться на відносній відстані не більше $2^{-d}(1-\varepsilon)$ від булевої функції n змінних, що задається вектором значень, $1\le d\le k<n,\ \varepsilon\in(0,1)$. Запропонований метод є більш ефективним в порівнянні з найкращим раніше відомим, що базується на алгоритмі Гопалана [24] (у певних випадках – в 1000 та більше разів), і може бути застосований на

практиці при аналізі кореляційних властивостей функцій ускладнення синхронних потокових шифрів при малих значеннях k і d, якщо кількість "відносно великих" за модулем коефіцієнтів Уолша-Адамара функції f не перевищує 20 (див. табл. 3.1).

Зменшення трудомісткості запропонованого методу в порівнянні з методом [24] досягається за рахунок застосування більш точної оцінки кількості шуканих наближень вхідної функції (твердження 3.1), а також більш економної організації обчислень, яка базується на детальному аналізі структури цих наближень (твердження 3.3 – 3.5).

2. Як і алгоритм Гопалана, алгоритми 3.1 – 3.4 методу можуть бути використані для побудови k-вимірних наближень булевих функцій, що задаються за допомогою оракулів (а не тільки векторів значень). В цьому випадку на першому етапі алгоритмів 3.1 і 3.2 замість швидкого перетворення Адамара слід застосовувати один з відомих швидких алгоритмів побудови високоймовірних лінійних наближень вхідної функції, наприклад, вдосконалений алгоритм Левіна [134, 135].

3. Результати практичного застосування запропонованого методу показують, що він дозволяє швидко знаходити високоймовірні наближення функцій від невеликої кількості змінних (наприклад, для функції Карле-Фенга від $n = 5$ змінних час виконання алгоритму 3.4 при $k \le 4$ складає декілька секунд, див. табл. 3.4).

4. Другим науковим результатом розділу є метод обчислення значень нижніх меж відносної відстані між зрівноваженою булевою функцією та множиною всіх k-вимірних функцій від n змінних. Зазначений метод запропоновано вперше і полягає у статистичному оцінюванні відносної відстані за допомогою спеціально розробленого ймовірнісного алгоритму, складність якого залежить лінійно від n та поліноміально від величин, обернених до точності та імовірності помилки алгоритму. Показано, що при малих значеннях k запропонований метод може бути ефективно використаний на практиці для

обґрунтування стійкості функцій ускладнення синхронних потокових шифрів відносно статистичних атак, наведених в розділі 2.

5. Результати практичного застосування запропонованого методу до функції ускладнення шифру Achterbahn-80 показують, що відносна відстань між нею та множиною 2-вимірних функцій є не менше ніж $0{,}4369$ з достовірністю не менше $0{,}75$, в той час як точне значення цієї відстані дорівнює $0{,}4375$. При цьому для оцінювання відносної відстані за допомогою запропонованого методу потрібно майже в 5 разів менше часу в порівнянні з алгоритмом обчислення її точного значення.

Основні наукові результати розділу опубліковані в [31, 33].

РОЗДІЛ 4

МЕТОД ПОШУКУ АЛГЕБРАЇЧНО ВИРОДЖЕНИХ НАБЛИЖЕНЬ БУЛЕВИХ ФУНКЦІЙ ДЛЯ ОЦІНЮВАННЯ СТІЙКОСТІ СИНХРОННИХ ПОТОКОВИХ ШИФРІВ ВІДНОСНО УЗАГАЛЬНЕНОЇ СТАТИСТИЧНОЇ АТАКИ

В другому розділі дисертації описано статистичну атаку на синхронні потокові шифри, яка узагальнює раніше відомі атаки на основі підібраних векторів ініціалізації [9, 10, 25 – 28]. Відзначено, що оцінювання стійкості синхронних потокових шифрів відносно цієї атаки зводиться до знаходження або обґрунтування відсутності певних алгебраїчно вироджених наближень булевих функцій, заданих за допомогою оракулів. Зазначена задача є обчислювально складною, а ефективні алгоритми її розв’язання відомі лише для окремих випадків [10, 23, 24, 29].

В даному розділі пропонується метод розв’язання поставленої задачі, який відрізняється за сутністю від відомих [10, 23, 24, 29] і базується на отриманих аналітичних умовах, яким задовольняють шукані наближення. На відміну від [23, 24], запропонований метод не має передумовою виконання будь-яких обмежень стосовно відстані, на якій треба відшукати наближення, а на відміну від [10, 29] він дозволяє знаходити наближення з більш широкого класу булевих функцій. Крім того, за певних умов запропонований метод надає можливість переконуватися у відсутності зазначених наближень, що дозволяє використовувати його для обґрунтування практичної стійкості синхронних потокових шифрів відносно узагальненої статистичної атаки.

4.1. Сутність, основні етапи і теоретичне обґрунтування методу, що пропонується

4.1.1. Означення основних понять та допоміжні результати. Нижче використовуються такі позначення:

$\#U$ – потужність множини U;

$<U>$ – підпростір векторного простору V_n, породжений множиною $U \subseteq V_n$;

$U^{\perp}$ – підпростір, дуальний до множини $U \subseteq V_n$: $U^{\perp} = \{\alpha \in V_n \mid \forall x \in U : \alpha x = 0\}$;

$\mathbf{F}_2^{m \times n}$ – множина матриць розміру $m \times n$ над полем $\mathbf{F}_2$;

I_m – одинична матриця порядку m над полем $\mathbf{F}_2$;

$\overline{a,b} = \{i \in \mathbf{Z} : a \le i \le b\}$, $a, b \in \mathbf{Z}$.

Позначимо B_n множину булевих функцій від n змінних. Нагадаємо, що відносна відстань між функціями $f, g \in B_n$ визначається за формулою $d(f,g) = 2^{-n} \#\{x \in V_n : f(x) \ne g(x)\}$, а відносна відстань між функцією $f \in B_n$ та множиною $U \subseteq B_n$ – за формулою $d(f,U) = \min\limits_{g \in U} d(f,g)$. Число $wt(f) = 2^{-n} \#\{x \in V_n : f(x) = 1\}$ називається відносною вагою функції $f \in B_n$.

Для будь-якої функції $F \in B_n$ покладемо

$$\hat{F}(\alpha) = 2^{-n} \sum_{x \in V_n} (-1)^{F(x) \oplus \alpha x}, \ \alpha \in V_n, \tag{4.1}$$

$$D_\alpha F(x) = F(x \oplus \alpha) \oplus F(x), \ x \in V_n. \tag{4.2}$$

Для будь-якого $m \in \overline{0,n}$ позначимо $L_{n,m}$ множину всіх m-вимірних підпросторів векторного простору V_n. Покладемо

$$\omega_F(H) = \sum_{\beta \in H} \hat{F}(\beta)^2, \; H \in L_{n,m}. \tag{4.3}$$

Справедливі такі рівності (див., наприклад, теорему 2.89 в [86] та "факт 11" в [23]):

$$\omega_F(H) = 1 - 2 \cdot \left(2^{-(n-m)} \sum_{\alpha \in H^{\perp}} wt(D_\alpha F) \right), \; H \in L_{n,m}. \tag{4.4}$$

$$wt(D_\alpha F) = \sum_{\beta \in V_n : \, \alpha\beta = 1} \hat{F}(\beta)^2, \; \alpha \in V_n. \tag{4.5}$$

Нагадаємо, що функція $g \in B_n$ називається m-вимірною, $m \in \overline{0,n}$, якщо вона допускає представлення у вигляді

$$g(x) = \varphi(xM), \; x \in V_n, \tag{4.6}$$

де $\varphi \in B_m$, $M \in \mathrm{F}_2^{n \times m}$. Булева функція від n змінних, яка є m-вимірною для деякого $m \le n-1$, називається алгебраїчно виродженою [23, 24, 29, 143].

Для будь-якого $H \in L_{n,m}$ позначимо $B_n(H)$ множину всіх функцій вигляду (4.6) таких, що $\varphi \in B_m$, $M \in \mathbf{F}_2^{n \times m}$ і стовпці матриці M належать підпростору H. Зрозуміло, що множина m-вимірних функцій від n змінних є об'єднанням множин $B_n(H)$ за всіма $H \in L_{n,m}$. Наступне твердження є прямим наслідком лем 3 і 4 в [143].

Твердження 4.1. [143]**.** Для будь-яких $F \in B_n$, $H \in L_{n,m}$ справедливі такі співвідношення:

$$d(F, B_n(H)) = 1/2 \cdot (1 - l_F(H)), \tag{4.7}$$

$$1/2 \cdot (1 - \sqrt{\omega_F(H)}) \le d(F, B_n(H)) \le 1/2 \cdot (1 - \omega_F(H)), \tag{4.8}$$

Де

$$l_F(H) = 2^{-m} \sum_{y \in V_m} \left| 2^{-(n-m)} \sum_{x \in L_y} (-1)^{F(x)} \right|, \tag{4.9}$$

$L_y = \{x \in V_n : xM = y\}$, $y \in V_m$.

За означенням [143] підпростір $H \in L_{n,m}$ називається θ-допустимим для функції $F \in B_n$, якщо виконується умова $d(F, B_n(H)) \le 1/2 \cdot (1 - \theta)$, $\theta \in (0,1)$. З твердження 4.1 випливає, що H є θ-допустимим підпростором для функції F тоді й тільки тоді, коли $l_F(H) \ge \theta$; крім того, H є θ-допустимим підпростором за умови $\omega_F(H) \ge \theta$.

4.1.2. Постановка задачі та основні етапи методу її розв'язання. Нехай $F = F(k, c)$ – булева функція від $n = l_0 + l_1$ змінних $k \in V_{l_0}$, $l \in V_{l_1}$, задана за допомогою оракула. Треба з'ясувати, чи існують для заданих чисел $s \in \overline{1, l_0 - 3}$, $\theta \in (0,1)$ функція $\varphi \in B_{s+l_1}$ та матриця $M_0 \in \mathrm{F}_2^{l_0 \times s}$ рангу s такі, що відносна відстань між функціями F та $\varphi(kM_0, c)$, $k \in V_{l_0}$, $l \in V_{l_1}$ не перевищує $1/2 \cdot (1 - \theta)$. У випадку існування зазначеної пари (M_0, φ) треба побудувати в явному вигляді матрицю M_0 та визначити алгоритм, що

реалізує функцію φ.

Назвемо функцію $g(k,c)=\varphi(kM_0,c)$, $k\in V_{l_0}$, $l\in V_{l_1}$, θ-*допустимим наближенням* функції F, якщо $\varphi\in B_{s+l_1}$, $M_0\in \mathrm{F}_2^{l_0\times s}$, $\mathrm{rank}(M_0)=s$ і $d(F,g)\le 1/2\cdot(1-\theta)$. Отже, задача полягає в перевірці існування та побудові (у випадку позитивного результату перевірки) хоча б одного $(s+l_1)$-вимірного θ-допустимого наближення функції F.

Введемо такі підпростори векторного простору V_n:

$$H_0=\{(0,c):c\in V_{l_1}\}, \tag{4.10}$$

$$L_0=H_0{}^{\perp}=\{(k,0):k\in V_{l_0}\}. \tag{4.11}$$

Метод розв'язання поставленої задачі, що пропонується, базується на такому твердженні, справедливість якого випливає безпосередньо з наведених означень.

Твердження 4.2. Нехай $F\in B_n$, $n=l_0+l_1$, $s\in\overline{1,l_0-3}$, $\theta\in(0,1)$ і $g(k,c)=\varphi(kM_0,c)$, $k\in V_{l_0}$, $l\in V_{l_1}$ є $(s+l_1)$-вимірним θ-допустимим наближенням функції F. Тоді стовпці матриці

$$M=\begin{pmatrix} M_0 & 0\\ 0 & I_{l_1}\end{pmatrix} \tag{4.12}$$

утворюють базис θ-допустимого для функції F підпростіру H такого, що $\dim H=s+l_1$ і $H\supseteq H_0$.

Навпаки, якщо $H\supseteq H_0$ є $(s+l_1)$-вимірним θ-допустимим для F підпростором, то існують базис цього підпростору, який складається зі стовпців

матриці M вигляду (4.12), де $\mathrm{rank}(M_0) = s$, а також функція $\varphi \in B_{s+l_1}$ такі, що $g(k,c) = \varphi(kM_0, c)$, $k \in V_{l_0}$, $l \in V_{l_1}$ є $(s+l_1)$-вимірним θ-допустимим наближенням функції F.

Отже, на підставі твердження 4.2 **задачу пошуку (обґрунтування відсутності) зазначених наближень функції F пропонується розв'язувати в два етапи.** На першому етапі здійснюється пошук $(s+l_1)$-вимірних θ-допустимих для функції F підпросторів $H \supseteq H_0$ (при цьому відсутність зазначених просторів свідчить про відсутність відповідних наближень). Потім, на другому етапі визначається алгоритм, що реалізує шукану функцію $\varphi \in B_{s+l_1}$.

Зауважимо, що для побудови функцій φ на другому етапі методу можна використовувати алгоритм 2.3. Тому далі зосередимо увагу на розв'язанні окремих задач першого етапу методу.

4.1.3. Алгоритм реалізації методу та його теоретичне обґрунтування. Встановимо необхідні умови існування $(s+l_1)$-вимірних θ-допустимих для функції F підпросторів $H \supseteq H_0$.

Для будь-якого підпростору $L \subseteq V_n$ покладемо $\xi_L(\alpha) = wt(D_\alpha F)$, $\alpha \in L$. Задамо на підпросторі L рівномірний розподіл ймовірностей. Тоді ξ_L є випадковою величиною. Наступна лема встановлює основні властивості цієї випадкової величини.

Лема 4.1. Справедливі такі твердження:

1) ξ_L є сумою попарно незалежних випадкових величин: $\xi_L(\alpha) = \sum_{\beta \in V_n \setminus L^\perp} \hat{F}(\beta)^2 I_\beta(\alpha)$, де $I_\beta(\alpha)$ – індикатор події $\{\alpha\beta = 1\}$, $\alpha \in L$;

2) $\mathbf{E}\xi_L = 1/2 \cdot (1 - \omega_F(L^\perp))$;

3) $\mathbf{D}\xi_L \le 1/4 \cdot \sum_{\beta \in V_n} \hat{F}(\beta)^4$;

4) за умови $\sum_{\beta \in V_n} \hat{F}(\beta)^4 \le \mu^2 < 1$ для будь-якого $\varepsilon > 0$ має місце нерівність

$$\mathbf{P}\{|\, \xi_L - \mathbf{E}\xi_L \,| \ge \varepsilon\} \le \left(\frac{\mu}{2\varepsilon} \right)^2 .$$

Доведення. Справедливість п. 1) випливає з формули (4.5) та незалежності подій $\{\alpha\beta_1 = 1\}$ і $\{\alpha\beta_2 = 1\}$ для будь-яких різних векторів $\beta_1, \beta_2 \in V_n \setminus L^{\perp}$; п. 2) є безпосереднім наслідком формул (4.3), (4.4), а п. 3) випливає з п. 1). Нарешті, п. 4) є наслідком п. 3) та нерівності Чебишова. Лему доведено.

Наступне твердження відіграє ключову роль для побудови методу пошуку допустимих підпросторів.

Твердження 4.3. Нехай H є $(s + l_1)$-вимірним θ-допустимим для функції F підпростором і $L = H^{\perp}$. Тоді для кожного $\alpha \in L$ справедлива нерівність $wt(D_\alpha F) \le 1 - \theta$. Крім того, за умови $\sum_{\beta \in V_n} \hat{F}(\beta)^4 \le \mu^2 < 1$ для будь-якого $\varepsilon > 0$ існує не менше ніж $2^{l_0 - s}\left(1 - \left(\frac{\mu}{2\varepsilon}\right)^2\right)$ векторів $\alpha \in L$, кожен з яких задовольняє нерівності $wt(D_\alpha F) \le 1/2 \cdot (1 - \theta^2) + \varepsilon$.

Доведення. Справедливість першої частини твердження випливає з теореми 2 в [143]. Для доведення другої частини розглянемо зазначену вище випадкову величину ξ_L. З п. 2) леми 4.1, умови $d(F, B_n(H)) \le 1/2 \cdot (1 - \theta)$ та нижньої межі (4.8) випливає, що $\mathbf{E}\xi_L \le 1/2 \cdot (1 - \theta^2)$. Звідси, використовуючи п. 4) леми 4.1 отримаємо, що при випадковому рівноймовірному виборі вектора α з підпростору L справедливі такі співвідношення:

$$\mathbf{P}\{wt(D_\alpha F) > 1/2 \cdot (1 - \theta^2) + \varepsilon\} = \mathbf{P}\{\xi_L - \mathbf{E}\xi_L > 1/2 \cdot (1 - \theta^2) - \mathbf{E}\xi_L + \varepsilon\} \le$$

$$\leq \mathbf{P}\{\xi_L - \mathbf{E}\xi_L > \varepsilon\} \leq \left(\frac{\mu}{2\varepsilon}\right)^2.$$

Розглянемо множину $D = \{\alpha \in L : wt(D_\alpha F) \leq 1/2 \cdot (1-\theta^2) + \varepsilon\}$. На підставі викладених вище міркувань отримаємо, що

$$|D| = 2^{\dim L}(1 - \mathbf{P}\{wt(D_\alpha F) > 1/2 \cdot (1-\theta^2) + \varepsilon\}) \geq 2^{l_0 - s}\left(1 - \left(\frac{\mu}{2\varepsilon}\right)^2\right).$$

Отже, твердження повністю доведено.

Покладемо $\mathcal{D}(F, \varepsilon) = \{\alpha \in L_0 : wt(D_\alpha F) \leq \varepsilon\}$, де підпростір L_0 визначається за формулою (4.11), $\varepsilon \in (0,1)$. Для будь-яких $\theta, \mu \in (0,1)$ позначимо

$$w(\theta, \mu) = \min\{1-\theta, 1/2 \cdot (1-\theta^2) + \mu\} \qquad (4.13)$$

Наслідок 4.1. Нехай за умови твердження 4.3 підпростір H містить підпростір H_0 вигляду (4.10). Тоді існує множина $D \subseteq \mathcal{D}(F, w(\theta, \mu))$ потужності $|D| \geq 3/4 \cdot 2^{l_0 - s}$ така, що $H \subseteq D^\perp$.

Отже, будь-який шуканий підпростір H складається с векторів, кожен з яких є ортогональним певній множині $D \subseteq \mathcal{D}(F, w(\theta, \mu))$ потужності не менше ніж $3/4 \cdot 2^{l_0 - s}$. Метод пошуку допустимих підпросторів, що пропонується, полягає в знаходженні зазначених векторів, побудові за ними матриці M вигляду (4.12) та оцінюванні відносної відстані між функцією F та множиною функцій вигляду (4.6).

Більш докладно, **алгоритм реалізації методу пошуку θ-допустимих підпросторів для функції F складається з таких кроків.**

1. Обчислити значення верхньої межі μ^2 параметра $\sum_{\beta \in V_n} \hat{F}(\beta)^4$.

2. Перевірити існування лінійно незалежних векторів $\alpha_1, \ldots, \alpha_{l_0 - s}$, що належать множині $\mathcal{D}(F, w(\theta, \mu))$, та побудувати ці вектори (у випадку позитивного результату перевірки).

3. Задати M як матрицю вигляду (4.12), стовпці якої утворюють базис векторного простору $< \alpha_1, \ldots, \alpha_{l_0 - s} >^{\perp}$.

4. Оцінити значення параметра $d(F, B_n(H))$ для підпростору H, що породжується стовпцями матриці M. Якщо $d(F, B_n(H)) \le 1/2 \cdot (1 - \theta)$, вважати H шуканим підпростором; в протилежному випадку зробити висновок про відсутність θ-допустимих для функції F підпросторів вимірності $s + l_1$.

Опишемо зараз (ймовірнісні) алгоритми, які використовуються на кроках 1, 2 і 4 та з'ясуємо, за яких умов запропонований метод дозволяє розв'язувати поставлену задачу з заданою достовірністю.

Для побудови алгоритму оцінювання параметра $\sum_{\beta \in V_n} \hat{F}(\beta)^4$ скористаємося відомою формулою [162]

$$\sum_{\beta \in V_n} \hat{F}(\beta)^4 = 2^{-3n} \sum_{x, y, z \in V_n} (-1)^{F(x) \oplus F(x \oplus y) \oplus F(x \oplus z) \oplus F(x \oplus y \oplus z)}, \qquad (4.14)$$

а також наступною лемою, яка неодноразово використовується далі.

Лема 4.2. [163]**.** Нехай $\zeta_1, \ldots, \zeta_t$ є незалежними випадковими величинами такими, що $\alpha_j \le \zeta_j \le \beta_j$, $\alpha_j, \beta_j \in \mathbf{R}$, $j \in \overline{1, t}$. Тоді для будь-якого $x > 0$

$$\mathbf{P}\left\{t^{-1}\sum_{l=1}^{t}\zeta_l - \mathbf{E}\left(t^{-1}\sum_{l=1}^{t}\zeta_l\right) \geq x\right\} \leq \exp\left\{-\frac{2t^2x^2}{\sum_{l=1}^{t}(\beta_l-\alpha_l)^2}\right\}.$$

Ймовірнісний алгоритм обчислення значень верхніх меж параметра $\sum_{\beta\in V_n}\hat{F}(\beta)^4$ наведено на рис. 4.1.

Алгоритм 4.1

Вхідні дані: функція $F \in B_n$, задана за допомогою оракула ($n = l_0 + l_1$); числа $\varepsilon, \delta \in (0,1)$.

1. Покласти $t = \left\lceil 2\varepsilon^{-2}\ln(\delta^{-1})\right\rceil$;

2. Згенерувати t незалежних випадкових елементів (X_i, Y_i, Z_i), де X_i, Y_i та Z_i є незалежними випадковими векторами з рівномірним розподілом ймовірностей на множині V_n, $i \in \overline{1,t}$;

3. Обчислити $\mu^2 = t^{-1}\sum_{i=1}^{t}(-1)^{F(X_i)\oplus F(X_i\oplus Y_i)\oplus F(X_i\oplus Z_i)\oplus F(X_i\oplus Y_i\oplus Z_i)}$.

Результат: число $\mu^2 + \varepsilon$.

Рис.4.1. Алгоритм, що виконується на першому кроці методу пошуку допустимих підпросторів

Безпосередньо з опису алгоритму 4.1, формули (4.14) та леми 4.2 випливає такий результат.

Твердження 4.4. Для будь-яких $\varepsilon, \delta \in (0,1)$ справедлива нерівність

$\mathbf{P}\{\sum_{\beta\in V_n}\hat{F}(\beta)^4 \le \mu^2\} \ge 1-\delta$. При цьому часова складність алгоритму 4.1 складає $t = O(\varepsilon^{-2}\ln\delta^{-1})$.

Опишемо зараз алгоритм, який виконується на другому кроці методу, що пропонується (рис. 4.2).

Алгоритм 4.2

Вхідні дані: функція $F \in B_n$, задана за допомогою оракула ($n = l_0 + l_1$); числа $s \in \overline{1, l_0 - 3}$, $\theta \in (0,1)$, $\mu \in (0,1)$, $\varepsilon \in (0,\theta)$, $\delta \in (0,1)$.

1. Обчислити $w(\theta,\mu)$ за формулою (4.13); покласти $t = \left\lceil 3^{-1}2^{s+3}\ln(2\delta^{-1})\right\rceil$, $l = \left\lceil 2^{-1}\varepsilon^{-2}\ln(2^{-s+1}t\delta^{-1})\right\rceil$;

2. Покласти α рівним нульовому вектору довжини l_0;

3. Для кожного $i \in \overline{1,t}$:

– згенерувати випадковий вектор ξ_i з рівномірним розподілом на множині $L_0 \setminus \{0\}$, де L_0 визначається за формулою (4.11);

– згенерувати незалежні випадкові вектори $X_{i,1}, \ldots, X_{i,l}$ з рівномірним розподілом на множині V_n та обчислити значення $\Delta_l(\xi_i) = l^{-1}\sum_{j=1}^{l} D_{\xi_i}F(X_{i,j})$;

– якщо $\Delta_l(\xi_i) \le w(\theta,\mu) + \varepsilon$, покласти $\alpha = \alpha_i$ та закінчити роботу.

Результат: вектор $\alpha = \xi_i$, якщо $\Delta_l(\xi_i) \le w(\theta,\mu) + \varepsilon$; $\alpha = 0_{l_0}$, в іншому випадку.

Рис. 4.2. Алгоритм знаходження векторів із множини $\mathcal{D}(F, w(\theta,\mu))$

Результатом виконання алгоритму 4.2 є ненульовий вектор α, що дорівнює значенню ξ_i з найменшим номером $i \in \overline{1,t}$, для якого $\Delta_l(\xi_i) \le w(\theta,\mu)+\varepsilon$, або нульовий вектор, якщо таких i не знайшлося.

Покажемо, що алгоритм 4.2 дозволяє надійно знаходити вектори з множини $\mathcal{D}(F, w(\theta,\mu))\setminus\{0\}$ за умови існування θ-допустимих підпросторів для функції F.

Твердження 4.5. Нехай існує θ-допустимий для функції F підпростір H вимірності $s+l_1$. Тоді випадковий вектор α, отриманий за допомогою алгоритму 4.2, належить множині $\mathcal{D}(F, w(\theta,\mu))\setminus\{0\}$ з ймовірністю не менше ніж $1-\delta$. Крім того, часова складність алгоритму 4.2 дорівнює

$$lt = O(2^s \varepsilon^{-2} \ln(\delta^{-1}) \ln(\delta^{-1} \ln \delta^{-1})). \qquad (4.15)$$

Доведення. На підставі наслідку 4.1 множина $\mathcal{D}(F, w(\theta,\mu))$ містить множину D потужності $\lceil 3/4 \cdot 2^{l_0-s} \rceil$. При цьому, якщо $\alpha \notin \mathcal{D}(F, w(\theta,\mu))\setminus\{0\}$, то відбувається одна з двох подій: або жоден з випадкових векторів $\xi_1, \dots, \xi_t$ не потрапляє до множини D, або існує $i \in \overline{1,t}$ таке, що $\xi_i \in D$ і $\Delta_l(\xi_i) > w(\theta,\mu)+\varepsilon$. Отже, $\mathbf{P}\{\alpha \notin \mathcal{D}(F, w(\theta,\mu))\setminus\{0\}\} \le p_1 + p_2$, де

$$p_1 = \mathbf{P}\{\xi_1, \dots, \xi_t \notin D\},\ p_2 = \mathbf{P}\left\{\bigcup_{i=1}^{t} \{\xi_i \in D, \Delta_l(\xi_i) > w(\theta,\mu)+\varepsilon\}\right\}.$$

В силу означення випадкових векторів $\xi_1, \dots, \xi_t$ і параметра t, а також умови $s \in \overline{1, l_0-3}$ справедливі такі співвідношення:

$$p_1 = (1 - \mathbf{P}\{\xi_1 \in D\})^t \le \left(1 - \frac{|D|-1}{2^{l_0}-1}\right)^t \le \left(1 - \frac{3/4 \cdot 2^{l_0-s}-1}{2^{l_0}-1}\right)^t \le$$

$$\le (1 - 3 \cdot 2^{-s-3})^t \le \exp\{-3 \cdot 2^{-s-3} t\} \le \delta/2.$$

Далі, використовуючи незалежність випадкових векторів ξ_i, $X_{i,j}$ ($i \in \overline{1,t}$, $j \in \overline{1,l}$), лему 4.2, означення параметра l та умову $s \in \overline{1, l_0 - 3}$ отримаємо, що

$$p_2 \le \sum_{i=1}^{t} \mathbf{P}\{\xi_i \in D, \Delta_l(\xi_i) > w(\theta,\mu) + \varepsilon\} \le \sum_{i=1}^{t} \sum_{a \in D \setminus \{0\}} \mathbf{P}\{\xi_i = a, \Delta_l(a) > w(\theta,\mu) + \varepsilon\} =$$

$$= t \sum_{a \in D \setminus \{0\}} \frac{1}{2^{l_0}-1} \mathbf{P}\{\Delta_l(a) > w(\theta,\mu) + \varepsilon\} \le \frac{3/4 \cdot 2^{l_0-s}+1}{2^{l_0}-1} t \exp\{-2l\varepsilon^2\} \le$$

$$\le 2^{-s} t \exp\{-2l\varepsilon^2\} \le \delta/2.$$

Таким чином, на підставі отриманих співвідношень $\mathbf{P}\{\alpha \in \mathcal{D}(F, w(\theta,\mu)) \setminus \{0\}\} \ge 1 - (p_1 + p_2) \ge 1 - \delta$, що і треба було довести. Нарешті, формула (4.15) випливає безпосередньо з опису алгоритму 4.2.

Твердження доведено.

Наслідок 4.2. Нехай $\alpha_1, \ldots, \alpha_r$ є випадковими векторами, отриманими в результаті r-кратного застосування алгоритму 4.2 до вхідних даних F, s, θ, μ, ε, $\delta = r^{-1}\delta'$, де $\delta' \in (0,1)$. Тоді за умови твердження 4.5 справедлива нерівність $\mathbf{P}\{\alpha_1, \ldots, \alpha_r \in \mathcal{D}(F, w(\theta,\mu)) \setminus \{0\}\} \ge 1 - \delta'$.

Опишемо, нарешті, алгоритми оцінювання параметра $d(F, B_n(H))$ на кроці 4 методу, що пропонується. Перший алгоритм базується на рівності (4.7) і дозволяє отримувати двосторонні статистичні оцінки зазначеного параметра. Другий алгоритм базується на верхній межі (4.8) і дозволяє отримувати тільки верхні оцінки значення $d(F, B_n(H))$, проте вимагає значно менше часу обчислень.

Алгоритм 4.3

Вхідні дані: функція $F \in B_n$, задана за допомогою оракула ($n = l_0 + l_1$); матриця $M \in \mathbf{F}_2^{n \times m}$, стовпці якої утворюють базис підпростору H (m $s \in \overline{1, l_0 - 3}$); числа $\varepsilon, \delta \in (0, 1)$.

1. Покласти $l = \left\lceil 2\varepsilon^{-2} \ln(4\delta^{-1}) \right\rceil$, $t = \left\lceil 8\varepsilon^{-2} \ln(4l\delta^{-1}) \right\rceil$;

2. Згенерувати незалежні випадкові вектори $\xi_1, \xi_2, \ldots, \xi_l$ з рівномірним розподілом на множині V_m; для кожного $i \in \overline{1, l}$ згенерувати незалежні випадкові вектори $\eta_{i1}, \ldots, \eta_{it}$ з рівномірним розподілом на множині $L_{\xi_i} = \{x \in V_n : xM = \xi_i\}$ та обчислити значення

$$\lambda_{l,t} = l^{-1} \sum_{i=1}^{l} \left| t^{-1} \sum_{j=1}^{t} (-1)^{F(\eta_{ij})} \right|.$$

Результат: числа $1/2 \cdot (1 - \lambda_{l,t}) - \varepsilon$, $1/2 \cdot (1 - \lambda_{l,t}) + \varepsilon$.

Рис. 4.3. Ймовірнісний алгоритм обчислення значень верхньої та нижньої меж параметра $d(F, B_n(H))$

Твердження 4.6. Для будь-яких $\varepsilon, \delta \in (0, 1)$ справедлива нерівність

$$\mathbf{P}\{|\, d_F(B_{n,m}(H)) - 1/2 \cdot (1-\lambda_{l,t})\,| \geq \varepsilon\} \leq \delta. \qquad (4.16)$$

При цьому часова складність алгоритму 4.3 дорівнює $lt = 16 \cdot C\varepsilon^{-4} \ln(4\delta^{-1}) \ln(8\varepsilon^{-2}\delta^{-1} \ln(4\delta^{-1}))$, де $C = \text{const} \geq 1$.

Доведення. Помітимо, що

$$|\lambda_{l,t} - l_F(H)| \leq \left| l^{-1} \sum_{i=1}^{l} \left| t^{-1} \sum_{j=1}^{t} (-1)^{F(\eta_{ij})} \right| - l^{-1} \sum_{i=1}^{l} \left| 2^{-(n-m)} \sum_{x \in L_{\xi_i}} (-1)^{F(x)} \right| \right| +$$

$$+ \left| l^{-1} \sum_{i=1}^{l} \left| 2^{-(n-m)} \sum_{x \in L_{\xi_i}} (-1)^{F(x)} \right| - 2^{-m} \sum_{y \in V_m} \left| 2^{-(n-m)} \sum_{x \in L_y} (-1)^{F(x)} \right| \right| = \nu_1 + \nu_2 .$$

Отже,

$$\mathbf{P}\{|\, d_F(B_{n,m}(H)) - 1/2 \cdot (1-\lambda_{l,t})\,| \geq \varepsilon\} = \mathbf{P}\{|\, \lambda_{l,t} - l_F(H)\,| \geq \varepsilon\} \leq$$

$$\leq \mathbf{P}\{\nu_1 \geq \varepsilon/2\} + \mathbf{P}\{\nu_2 \geq \varepsilon/2\}. \qquad (4.17)$$

Оцінимо зверху другий доданок у правій частині нерівності (4.17). Позначимо $\varsigma_i = \left| 2^{-(n-m)} \sum_{x \in L_{\xi_i}} (-1)^{F(x)} \right|$, $i \in \overline{1,l}$. Випадкові величини $\varsigma_1, \varsigma_2, \ldots, \varsigma_l$ є незалежними, однаково розподіленими та приймають значення у проміжку $[0,1]$. При цьому $\mathbf{E}\varsigma_i = 2^{-m} \sum_{y \in V_m} \left| 2^{-(n-m)} \sum_{x \in L_y} (-1)^{F(x)} \right| = l_F(H)$, $i \in \overline{1,l}$ і $\nu_2 = \left| l^{-1} \sum_{i=1}^{l} \varsigma_i - l^{-1} \sum_{i=1}^{l} \mathbf{E}\varsigma_i \right|$. Отже, на підставі леми 4.2 та означення параметра l

справедлива нерівність

$$\mathbf{P}\{\nu_2 \geq \varepsilon/2\} \leq 2\exp\{-2l(\varepsilon/2)^2\} \leq \delta/2. \qquad (4.18)$$

Оцінимо зараз перший доданок у правій частині нерівності (4.17). Помітимо, що

$$\mathbf{P}\{\nu_1 \geq \varepsilon/2\} = \mathbf{P}\left\{\left\| l^{-1}\sum_{i=1}^{l}\left|t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_{ij})}\right| - l^{-1}\sum_{i=1}^{l}\left|2^{-(n-m)}\sum_{x\in L_{\xi_i}}(-1)^{F(x)}\right|\right\| \geq \varepsilon/2\right\} \leq$$

$$\leq \mathbf{P}\left\{\left\| l^{-1}\sum_{i=1}^{l}\left(t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_{ij})} - 2^{-(n-m)}\sum_{x\in L_{\xi_i}}(-1)^{F(x)}\right)\right\| \geq \varepsilon/2\right\} \leq$$

$$\leq \mathbf{P}\left\{\max_{1\leq i\leq l}\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_{ij})} - 2^{-(n-m)}\sum_{x\in L_{\xi_i}}(-1)^{F(x)}\right| \geq \varepsilon/2\right\} \leq$$

$$\leq \mathbf{P}\left\{\bigcup_{i=1}^{l}\left\{\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_{ij})} - 2^{-(n-m)}\sum_{x\in L_{\xi_i}}(-1)^{F(x)}\right| \geq \varepsilon/2\right\}\right\} \leq$$

$$\leq \sum_{i=1}^{l}\mathbf{P}\left\{\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_{ij})} - 2^{-(n-m)}\sum_{x\in L_{\xi_i}}(-1)^{F(x)}\right| \geq \varepsilon/2\right\}.$$

Далі, згідно з а формулою повної ймовірності,

$$\mathbf{P}\left\{\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_{ij})} - 2^{-(n-m)}\sum_{x\in L_{\xi_i}}(-1)^{F(x)}\right| \geq \varepsilon/2\right\} =$$

$$= 2^{-m}\sum_{y\in V_m}\mathbf{P}\left\{\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_{ij})} - 2^{-(n-m)}\sum_{x\in L_{\xi_i}}(-1)^{F(x)}\right| \geq \varepsilon/2 \;\middle|\; \xi_i = y\right\} =$$

$$2^{-m}\sum_{y\in V_m}\mathbf{P}\left\{\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_j)} - 2^{-(n-m)}\sum_{x\in L_y}(-1)^{F(x)}\right| \geq \varepsilon/2\right\},$$

де $\eta_1,\ldots,\eta_t$ є незалежними випадковими величинами з рівномірним розподілом ймовірностей на множині L_y. Отже, на підставі леми 4.2 маємо

$$\mathbf{P}\left\{\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_j)} - 2^{-(n-m)}\sum_{x\in L_y}(-1)^{F(x)}\right| \geq \varepsilon/2\right\} =$$

$$= \mathbf{P}\left\{\left| t^{-1}\sum_{j=1}^{t}(-1)^{F(\eta_j)} - t^{-1}\sum_{j=1}^{t}\mathbf{E}(-1)^{F(\eta_j)}\right| \geq \varepsilon/2\right\} \leq 2\exp\{-1/2\cdot t(\varepsilon/2)^2\},$$

звідки в силу означення параметра t випливає, що

$$\mathbf{P}\{\nu_1 \geq \varepsilon/2\} = 2l\exp\{-1/2\cdot t(\varepsilon/2)^2\} \leq \delta/2. \qquad (4.19)$$

Таким чином, на підставі формул (4.17) – (4.19) справедлива нерівність (4.16). Твердження доведено.

Опишемо другий алгоритм обчислення значень верхніх меж параметра

$d(F, B_n(H))$. Позначимо M' $(n-m)\times n$-матрицю, рядки якої утворюють базис дуального до H підпростору $H^{\perp}$. На підставі формул (4.4), (4.8) справедлива нерівність

$$d(F, B_n(H)) \le 2^{-(n-m)}\ 2^{-n} \sum_{(\alpha,\, x)\, \in\, H^{\perp}\times V_n} (F(x\oplus\alpha)\oplus F(x)). \qquad (4.20)$$

Алгоритм 4.4

Вхідні дані: функція $F \in B_n$, задана за допомогою оракула ($n = l_0 + l_1$); $(n-m)\times n$-матриця M', рядки якої утворюють базис підпростору $H^{\perp}$ ($m = s + l_1$, $s \in \overline{1, l_0 - 3}$); числа $\varepsilon, \delta \in (0,1)$.

1. Покласти $t = \left\lceil 2^{-1}\varepsilon^{-2}\ln(\delta^{-1}) \right\rceil$;

2. Згенерувати t незалежних випадкових елементів (X_i, Y_i), де X_i та Y_i є незалежними випадковими векторами з рівномірними розподілами ймовірностей на множинах V_n та V_{n-m} відповідно, $i \in \overline{1,t}$;

3. Обчислити значення $\Delta_t = t^{-1}\sum_{i=1}^{t}(F(X_i \oplus Y_i M') \oplus F(X_i))$.

Результат: число $\Delta_t + \varepsilon$.

Рис. 4.4. Ймовірнісний алгоритм обчислення значень верхніх меж параметра $d(F, B_n(H))$

Безпосередньо з опису алгоритму 4.4, нерівності (4.20) та леми 4.2 випливає такий результат.

Твердження 4.7. Для будь-яких $\varepsilon, \delta \in (0,1)$ справедлива нерівність $\mathbf{P}\{d_F(B_{n,m}(H)) \le \Delta_t + \varepsilon\} \ge 1 - \delta$. При цьому часова складність алгоритму 4.4

дорівнює $t = O(\varepsilon^{-2} \ln \delta^{-1})$.

Підведемо підсумок наведеним вище результатам. Запропонований метод пошуку (обґрунтування відсутності) θ-допустимих наближень функції F складається з двох етапів, основним з яких є перший, що полягає в пошуку (обґрунтуванні відстутності) θ-допустимих для функції F підпросторів. На другому етапі (за умови наявності шуканих підпросторів) здійснюється побудова функції $\varphi \in B_{s+l_1}$ за допомогою алгоритму 2.3.

Метод пошуку θ-допустимих підпросторів складається с чотирьох кроків, для виконання яких запропоновані поліноміальні ймовірнісні алгоритми 4.1 – 4.4. Сутність методу полягає у застосуванні отриманих аналітичних умов, яким задовольняють базисні вектори кожного з шуканих підпросторів (див. твердження 4.3 та наслідок 4.1). Зазначені умови дозволяють скоротити час пошуку θ-допустимих підпросторів у порівнянні з алгоритмом повного перебору. Зауважимо також, що загальна трудомісткість описаного методу суттєво залежить від кількості випадкових векторів, які формуються за допомогою алгоритму 4.2, і може бути надто великою, якщо множина $\mathcal{D}(F, w(\theta, \mu))$ має велику потужність. Поряд з тим, якщо результатом алгоритму 4.2 є нульовий вектор, то на підставі твердження 4.5 функція F не має $(s + l_1)$-вимірних θ-допустимих наближень з достовірністю не менше ніж $1 - \delta$. В цьому випадку трудомісткість методу визначається за формулою (4.15).

Нижче показано, що викладений метод може бути ефективно використаний на практиці як для виявлення вразливостей, так і для обгрунтування стійкості сучасних синхронних потокових шифрів.

4.2. Практичне застосування запропонованого методу до оцінювання стійкості шифрів Grain-128 та SNOW 2.0 відносно узагальненої статистичної атаки

4.2.1. Побудова узагальненої статистичної атаки на редуковану версію шифру Grain-128. Застосуємо описаний метод до функції $F = F_{\mathrm{FKM}}(k, c)$, $k = (k_0, \ldots, k_{127}) \in V_{128}$, $c = (c_0, \ldots, c_{95}) \in V_{96}$, яка використовується для реалізації узагальненої статистичної атаки на редуковану версію шифру Grain-128 (див. п. 2.2.3). Для цього знайдемо $(s + l_1)$-вимірне θ-допустиме наближення функції F при $s = 52$, $\theta = 0{,}57$.

Перш за все, обчислимо значення верхньої межі μ^2 параметра $\sum_{\beta \in V_n} \hat{F}(\beta)^4$, застосовуючи десять разів до функції F алгоритм 4.1 при $\varepsilon = 0{,}01$, $\delta = 0{,}1$ (табл. 4.1). Середнє арифметичне значення μ^2 за десятьма проведеними експериментами дорівнює $0{,}5105$. Час однократного виконання алгоритму 4.1 складає 43 хвилини.

Таблиця 4.1

Результати виконання алгоритму 4.1

№ експерименту	μ^2
1	0,5119
2	0,5096
3	0,5060
4	0,5087
5	0,5107
6	0,5132
7	0,5093
8	0,5117
9	0,5105
10	0,5136

Далі застосуємо $r = 600$ разів алгоритм 4.2 до вхідних даних F, $s = 52$, $\theta = 0{,}57$, $\mu = \sqrt{0{,}5105} = 0{,}7145$, $\varepsilon = 0{,}05$, $\delta = r^{-1}\delta'$, де $\delta' = 0{,}1$. В результаті отримаємо список випадкових векторів $\alpha_1, \ldots, \alpha_r$, які задовольняють умові

$\mathbf{P}\{\alpha_1, \ldots, \alpha_r \in \mathcal{D}(F, w(\theta,\mu)) \setminus \{0\}\} \geq 1-\delta'$ (див. наслідок 4.2). Параметр $w(\theta,\mu)$ в цьому випадку складає 0,43. В табл. 4.2 показано десять отриманих векторів та відповідні їм значення параметра $\Delta_l(\alpha)$ (див. опис алгоритму 4.2). Середній час однократного застосування алгоритму 4.2 складає приблизно 2 хвилини.

Таблиця 4.2

Перші десять векторів, отриманих за допомогою алгоритму 4.2

№ вектора α у списку	Шістнадцяткове значення α	$\Delta_l(\alpha)$	$\Delta_l(\alpha)+\varepsilon$
1	d2a380839522b94c506001c1351c7241	0,2170	0,2670
2	691e3c0a3bc85309cbb891213c367e4d	0,2196	0,2696
3	d7c4a397c0bc0400b0429993113a7b80	0,2410	0,2910
4	2925dec4f8ef5a593e9dfdc140c4c868	0,2178	0,2678
5	fe0a5f9c09e2c7ed255dca3a19eeda70	0,2479	0,2979
6	0a47ac5858e0f8e87e5a139a572477e8	0,2226	0,2726
7	8826bd9d0ca203826729064b5872056c	0,2068	0,2568
8	f4954e03dfd98b0c3026e332635ef7c8	0,2252	0,2752
9	c8afe96cc49e3d28542e6fc909c43378	0,2375	0,2875
10	f81a2ab7d2de82c4a85420710dc4dc4d	0,2192	0,2692

Далі, використовуючи перші $l_0 - s = 76$ векторів з отриманого списку, побудуємо матрицю M вигляду (4.12) як зазначено на кроці 3 запропонованого методу. На рис. 4.5 показано (у транспонованому вигляді) підматрицю M_0 отриманої матриці M.

Оцінимо зараз значення параметра $d(F, B_n(H))$, де H є підпростором, породженим стовпцями матриці M, застосовуючи алгоритми 4.3 і 4.4 до вхідних даних F, M, $\varepsilon = 0{,}2$, $\delta = 0{,}1$ (табл. 4.3).

Середні арифметичні значення оцінок параметра $d(F, B_n(H))$, отримані за допомогою алгоритмів 4.3 та 4.4 дорівнюють 0,3497 та 0,4307 відповідно. Звідси на підставі описів зазначених алгоритмів і тверджень 4.6 та 4.7 випливає,

що $d(F, B_n(H)) \le 0{,}3497$ з достовірністю не менше ніж $1-\delta/2-2(\delta/4)^{10} \ge 0{,}946$ і $d(F, B_n(H)) \le 0{,}4307$ з достовірністю не менше ніж $1-\delta^{10} = 1-10^{-10}$.

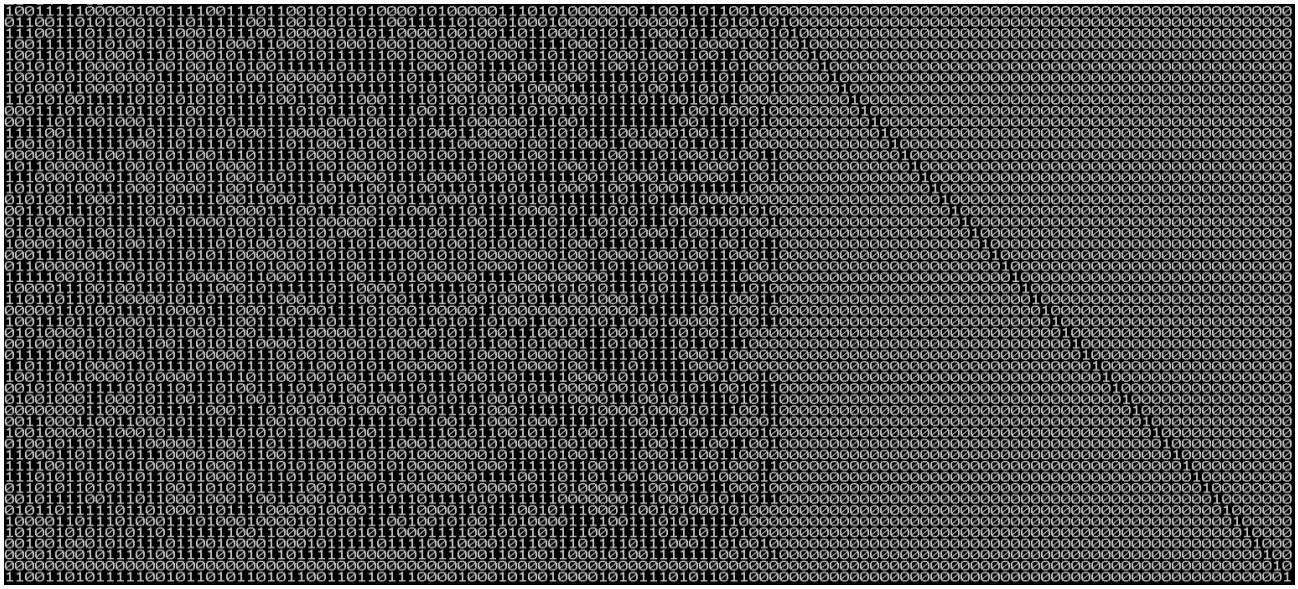

Рис. 4.5. Матриця, транспонована до M_0

Таблиця 4.3

Результати виконання алгоритмів 4.3 і 4.4

№ експерименту	Відносна відстань між функцією F та множиною $B_n(H)$					
	Алгоритм 4.3			Алгоритм 4.4		
	d	$d+\varepsilon$	T, хв	d	$d+\varepsilon$	T, с
1	0,1557	0,3557	150	0,1724	0,3724	2
2	0,1486	0,3486	150	0,2759	0,4759	2
3	0,1665	0,3665	150	0,1379	0,3379	2
4	0,1583	0,3583	150	0,2414	0,4414	2
5	0,1505	0,3505	150	0,2414	0,4414	2
6	0,1577	0,3577	150	0,3448	0,5448	2
7	0,1358	0,3358	150	0,1379	0,3379	2
8	0,1477	0,3477	150	0,2069	0,4036	2
9	0,1369	0,3369	150	0,3793	0,5793	2
10	0,1396	0,3396	150	0,1724	0,3724	2

Нарешті, скористаємося алгоритмом обчислення функції φ, описаним в другому розділі (див. алгоритм 2.3) для побудови наближення $g(k,c)=\varphi((k,c)M)$, $k\in V_{l_0}$, $l\in V_{l_1}$ функції F. В табл. 4.4 (з правого боку) показано результати оцінювання відносної відстані між функціями F та g з точністю $\varepsilon=0{,}005$ і достовірністю $1-\delta=0{,}9$. Для порівняння (з лівого боку) наведено верхні оцінки параметра $d(F,B_n(H))$, отримані за допомогою алгоритму 4.4 при тих самих значеннях ε і δ.

Таблиця 4.4

Статистичні оцінки параметра $d(F,B_n(H))$

№ експерименту	Відносна відстань між функцією F та множиною $B_n(H)$ (алгоритм 4.4)			Відносна відстань між функцією F та її запропонованим наближенням g		
	d	$d+\varepsilon$	T, хв	d	$d+\varepsilon$	T, с
1	0,2367	0,2417	41	0,1876	0,1926	40
2	0,2391	0,2441	41	0,1652	0,1702	40
3	0,2371	0,2421	41	0,2103	0,2153	40
4	0,2384	0,2434	41	0,3286	0,3336	40
5	0,2344	0,2394	41	0,2056	0,2106	40
6	0,2395	0,2445	41	0,2762	0,2812	40
7	0,2374	0,2424	41	0,2634	0,2684	40
8	0,2350	0,2400	41	0,1844	0,1894	40
9	0,2355	0,2405	41	0,1786	0,1836	40
10	0,2348	0,2398	41	0,1779	0,1829	40

Результати табл. 4.4, дозволяють отримати більш точні (в порівнянні з табл. 4.3) оцінки параметра $d_F(B_{n,m}(H))$. Так, середнє арифметичне значення оцінок цього параметра, обчислених за допомогою алгоритму 4.4, складає $0{,}2418$, а найменше (з десяти отриманих) значення параметра $d(F,g)$ дорівнює $0{,}1702$. Таким чином, $d(F,g)\le 0{,}1702$ з достовірністю не менше ніж $1-\delta=0{,}9$.

При цьому двійкове значення вектора z, який генерується на першому кроці алгоритму обчислення функції φ та визначає наближення g, є таким:

11010111011111100000001110010101111110000000110001100100100010101100100010100

Відзначимо, що алгоритми 4.1 – 4.4 та алгоритм обчислення функції φ реалізовано на платформі .NET Framework 4.0 (C#). Для виконання обчислень використано ПК на базі 64-розрядної операційної системи Windows 7 Service Pack 1 з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та оперативною пам'яттю обсягом 4,00 ГБ. Використовуючи отримане наближення функції F, можна побудувати узагальнену статистичну атаку на редуковану версію шифру Grain-128 [10]. Трудомісткість цієї атаки визначається за формулою (2.24) (див. с. 62 другого розділу дисертації):

$$T(l_0, s, r) = O\left(2^s r(T_F + T_\varphi) + s(l_0 - s)2^{l_0 - s} T_{\mathcal{A}}\right),$$

де T_F, T_φ і $T_{\mathcal{A}}$ є, відповідно, часові складності обчислення значень функцій F, φ і алгоритму $\mathcal{A}$ опробування ключів на другому етапі атаки, $r = \left\lceil 2^{2\nu+3} \ln\left(2^s \delta^{-1}\right)\right\rceil$, де число $\nu > 1$ визначається з нерівності $d(f, g) \le 1/2 - 2^{-\nu}$. Вважаючи $T_F = 128$, $T_\varphi = T_F n^2$, $T_{\mathcal{A}} = 3 \cdot 128$, та використовуючи оцінку $d(F, g) \le 0{,}1702$, отримаємо, що (за умови теореми 2.2, див. с. 64 другого розділу дисертації) трудомісткість узагальненої статистичної атаки на зазначену версію шифру Grain-128 не перевищує 2^{97}, що є в 2^{27} разів менше ніж трудомісткість раніше відомої атаки [10].

4.2.2. Обґрунтування практичної стійкості шифру SNOW-2.0 відносно узагальненої статистичної атаки. Нагадаємо, що SNOW-2.0 [5] – це слово-орієнтований синхронний потоковий шифр (довжина слова складає 32 біти), який на сьогодні є прототипом майбутнього національного стандарту потокового шифрування України. Довжина ключа SNOW-2.0 складає 128 або 256 біт (4 або 8 слів відповідно), а довжина вектора ініціалізації є 128 біт (4 слова). Шифр має дуже високу швидкодію, яка досягається використанням мінімального набору операцій (додавання слів за модулями 2 та 2^{32} відповідно, байтового зсуву слова та звернення до таблиці), які є доступними в сучасних процесорах.

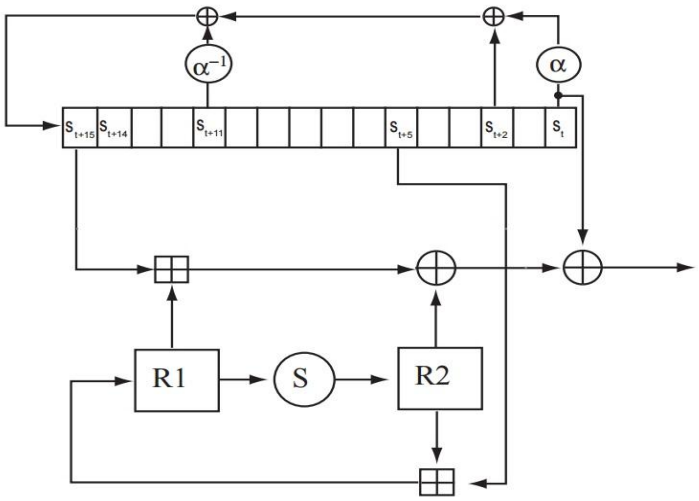

Рис. 4.6. Схема генератора гами шифру SNOW-2.0

Для оцінювання стійкості шифру SNOW-2.0 відносно узагальненої статистичної атаки необхідно задати функцію-оракул $F: V_{l_0} \times V_{l_1} \to \{0,1\}$, яка задовольняє умовам а) – в), зазначеним в розділі 2. Приймемо в ролі F функцію $F_{\text{SNOW-2.0}}$ з одним раундом реініціалізації (на відміну від повного алгоритму формування початкового стану, який складається з 32 раундів). Значення функції $F_{\text{SNOW-2.0}}$ на довільному наборі $(k,c) \in V_{l_0} \times V_{l_1}$ дорівнює молодшому біту слова шифрувальної гами, що виробляється за ключем k та

вектором ініціалізації c.

Нагадаємо, що необхідною умовою застосовності запропонованої атаки є існування θ-допустимих для функції $F_{\text{SNOW-2.0}}$ підпросторів помірної вимірності $m = s + l_1$ при достатньо великому значенні $\theta \in (0, 1)$. Для перевірки існування таких підпросторів скористаємося запропонованим вище методом.

Застосування методу дає негативний результат вже на етапі виконання алгоритму 4.2. Наприклад, при $\varepsilon = 0{,}01$, $\delta = 0{,}1$, $s = 20$ та $\theta = 0{,}6$ не виявляється жодного вектора α, який належить множині $\mathcal{D}(F, w(\theta, \mu))$, тобто задовольняє умові $wt(D_{(\alpha,0)}F_{\text{SNOW-2.0}}) \le 0{,}41$ (див. табл. 4.5, де показано декілька векторів α з отриманими значеннями параметра $w(\alpha) = wt(D_{(\alpha,0)}F_{\text{SNOW-2.0}})$). Час виконання алгоритму на комп'ютері з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам'яті 4 Гб RAM на базі 32-розрядної ОС Windows 7 Service Pack 1, складає приблизно 185 годин. Отриманий результат свідчить про відсутність (з достовірністю не менше $1 - \delta = 0{,}99$) наближень функції $F_{\text{SNOW-2.0}}$, які знаходяться від неї на відносній відстані не більше ніж $1/2 \cdot (1 - \theta) = 0{,}2$ та мають вигляд $\varphi(kM_0, c)$, $(k, c) \in V_{128} \times V_{128}$, де $\varphi : V_{148} \to \{0, 1\}$, $M_0 \in \mathbf{F}_2^{128\times 20}$.

Таблиця 4.5

Приклади векторів ξ_i, отриманих за допомогою алгоритму 4.2, застосованого до функції $F_{\text{SNOW-2.0}}$ (1 раунд)

№	$w(\xi_i)$	Шістнадцяткове представлення вектора ξ_i
1	0,4944	4f9d28167ed63f58546046ce37a89fe9
2	0,5009	702beb68257e915ace69a1831fe8699e
3	0,5023	f311719d23cd5db7b1fd11275b6f1649
4	0,5029	88bab9cfed5d4adf5986e6cf6432b99e
5	0,5021	22ced8d46b1c37d04df0a3f0ff903c88
6	0,4984	993abe8e37117cc41f85d12644f72836
7	0,4977	741b9f6460f138ac2456f22d7b974046
8	0,5029	92bef7ad8f6fbcb75b55a05b8b9bb1ad

Поряд з тим, подальше експериментальне дослідження функції $F_{\text{SNOW-2.0}}$ дозволяє знайти вектори, які задовольняють умові $wt(D_{(\alpha,0)}F_{\text{SNOW-2.0}}) \le 0{,}41$. Для знаходження таких векторів необхідно оцінити вплив окремих байтів ключа на вихідне значення функції $F_{\text{SNOW-2.0}}$. Іншими словами необхідно виконати таку процедуру.

1. Покласти α рівним нульовому вектору довжини l_0.

2. Для кожного окремого байту $byte \in \overline{0,15}$ вектора α:

- згенерувати випадкове число $rand$ від 0 до 255 та присвоїти $a[byte] = rand$;

- оцінити значення відносної ваги $\Delta_l(\alpha)$;

- якщо виконується умова $\Delta_l(\alpha) \le wt(D_{(\alpha,0)}F_{RC4}) \le 0{,}41$, закінчити роботу та повернути отриманий вектор α; в протилежному випадку покласти $a[byte] = 0$ та перейти до наступного байту ($byte++$);

В результаті застосування цієї процедури до функції $F_{\text{SNOW-2.0}}$ знайдені наступні позиції “слабких” байтів ключа: 1, 3, 5, 6, 12, 14, 15. Застосування зазначеної процедури до функції $F_{\text{SNOW-2.0}}$ з двома та трьома раундами реініціалізації дозволило знайти позиції “слабких” байтів ключа, які дорівнюють 0, 1, 2 та 14 відповідно. При подальшому збільшенні числа раундів реініціалізації (починаючи з 4 та вище) виявлений ефект повністю нівелюється.

Зазначимо, що негативний результат застосування запропонованого методу до $F_{\text{SNOW-2.0}}$ з одним раундом реініціалізації обумовлено дуже малою ймовірністю $(2^{56} \cdot 2^{-128} = 2^{-72})$ генерування випадкового вектора ξ_i спеціального виду, який можна отримати за допомогою наведеної вище процедури (нагадаємо, що метод є практично застосовним, якщо “низько-вагові” вектори складають як мінімум 75% всієї множини векторів, які перебираються, та якщо параметр s є не надто великим).

Нарешті, наведемо результати оцінювання стійкості повної версії шифру

SNOW-2.0 відносно узагальненої статистичної атаки. Приймемо в ролі F функцію $F_{\text{SNOW-2.0}}$ з повним алгоритмом формування початкового стану, який складається з 32 раундів, та застосуємо запропонований метод для перевірки існування θ-допустимих для функції $F_{\text{SNOW-2.0}}$ підпросторів.

Застосування методу дає негативний результат на етапі виконання алгоритму 4.2. При аналогічних значеннях ε, δ, s та θ не виявляється жодного вектора α, який належить множині $\mathcal{D}(F, w(\theta,\mu))$: в табл. 4.6 показано приклади відповідних векторів. Час виконання алгоритму на комп'ютері (з параметрами, що зазначені вище) складає приблизно 187 годин. Отриманий результат свідчить про відсутність (з достовірністю не менше $1-\delta=0{,}99$) наближень функції $F_{\text{SNOW-2.0}}$, які знаходяться від неї на відносній відстані не більше ніж $1/2\cdot(1-\theta)=0{,}2$ та мають вигляд $\varphi(kM_0, c)$, $(k,c)\in V_{128}\times V_{128}$, де $\varphi: V_{148}\to\{0,1\}$, $M_0\in\mathbf{F}_2^{128\times 20}$.

Таблиця 4.6 – приклади векторів ξ_i, отриманих за допомогою алгоритму 4.2, застосованого до функції $F_{\text{SNOW-2.0}}$ (32 раунда)

№	$w(\xi_i)$	Шістнадцяткове представлення вектора ξ_i
1	0,4999	3da7ce4ac273d318d36f3a40651a6f4
2	0,5012	d25c23adaaed16525cc62c6ec54553f
3	0,5077	f4181d392183d96f925849f6815ef2e
4	0,4973	1f9c9fd17564c215e3a526d1ed81560
5	0,5010	da7aac062d33fd5ac7cf05c879d220
6	0,5042	dcb78384c4585f6974bfb48930dc11e2
7	0,5015	1660f128325ccb4c9bacbfb3464514e
8	0,5039	acf81891b04f14a992235493de26f320

Застосування алгоритму 4.2 до деяких інших функцій F, що задаються за допомогою шифру SNOW-2.0, приводить до аналогічних результатів: при $s\le 20$, $\theta=0{,}6$ виявляється неможливим побудувати θ-допустимий для функції

F підпростір вимірності $s+l_1$, а отже, і реалізувати узагальнену статистичну атаку.

Висновки

1. Основним результатом розділу є метод знаходження (обґрунтування відсутності) певних алгебраїчно вироджених наближень булевих функцій, заданих за допомогою оракулів. Наявність таких наближень є необхідною умовою застосовності узагальненої статистичної атаки на синхронні потокові шифри, описаної в розділі 2, в той час, як їх відсутність гарантує практичну стійкість синхронного потокового шифру відносно цієї атаки.

2. Запропонований метод відрізняється за сутністю від раніше відомих [10, 23, 24, 29] та базується на отриманих необхідних умовах існування шуканих наближень. На відміну від [23, 24], він не має передумовою виконання будь-яких обмежень стосовно відстані, на якій треба відшукати наближення, а на відміну від [10, 29] він дозволяє знаходити наближення з більш широкого класу булевих функцій. Крім того, за певних умов запропонований метод надає можливість переконуватися у відсутності зазначених наближень, що дозволяє використовувати його для обґрунтування практичної стійкості синхронних потокових шифрів відносно статистичних атак.

3. Метод складається з двох етапів, на першому з яких здійснюється пошук певних підпросторів, що є допустимими для вхідної функції-оракула (при цьому відсутність зазначених просторів свідчить про відсутність відповідних наближень). На другому етапі за знайденим підпростором будується булева функція від меншої кількості змінних, яка (поряд з базисом знайденого підпростору) задає наближення вхідної функції. Для розв’язання окремих задач на кожному етапі методу запропоновано поліноміальні ймовірнісні алгоритми, які можуть бути застосовані на практиці до функцій-

оракулів від декількох десятків чи сотень змінних.

4. Застосування розробленого методу до функції F_{FKM}, яка використовується для побудови узагальненої статистичної атаки на редуковану версію шифру Grain-128, дозволяє отримати $(s+l_1)$-вимірне θ-допустиме наближення цієї функції при $s=52$, $\theta=0{,}57$ за 53 години на комп'ютері з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам'яті 4 Гб RAM на базі 32-розрядної ОС Windows 7 Service Pack 1 (сумарний час виконання алгоритмів 4.1 (10 разів), 4.2 (600 разів), 4.3 (10 разів) та оцінювання відносної відстані між F_{FKM} та спеціальним наближенням побудованим за алгоритмом 4.5). Отримане наближення дозволяє реалізувати атаку на зазначену версію шифру, складність якої (за умови теореми 2.1) не перевищує 2^{97}, в той час як складність раніше відомої атаки [10] дорівнює 2^{124}.

5. Застосування розробленого методу до низки функцій $F_{\mathrm{SNOW\text{-}2.0}}$ для побудови узагальненої статистичної атаки на шифр SNOW-2.0 дає негативний результат. При $s\le 20$, $\theta=0{,}6$ з достовірністю не менше ніж 0,99 жодна з зазначених функцій не має наближень, які знаходяться від неї на відносній відстані не більше ніж $1/2\cdot(1-\theta)=0{,}2$ та мають вигляд $\varphi(kM_0,c)$, $(k,c)\in V_{128}\times V_{128}$, де $\varphi: V_{148}\to\{0,1\}$, $M_0\in\mathbf{F}_2^{128\times 20}$. Це свідчить про практичну стійкість шифру відносно узагальненої статистичної атаки.

Основні наукові результати розділу опубліковані в [35, 44].

ВИСНОВКИ

Важливою задачею інформаційної безпеки держави є створення та підтримання умов, які гарантують безпечну обробку, зберігання та передачу державних інформаційних ресурсів. Для вирішення цієї задачі, поряд з технічними та організаційними, застосовують криптографічні методи, які базуються на спеціальних математичних перетвореннях інформації, що захищається. Важливою складовою забезпечення безпеки інформації в сучасних спеціальних інформаційно-телекомунікаційних системах є синхронні потокові шифри. Це обумовлено, головним чином, надвисокою швидкістю потокового шифрування на канальному, мережевому та транспортному рівнях.

Значний інтерес до потокових шифрів спостерігається у європейському науковому криптографічному співтоваристві, про що свідчить проведення міжнародних конкурсів NESSIE та eSTREAM, спрямованих на створення стійких потокових шифрів, придатних до широкого застосування. Через відсутність на даний час національного стандарту потокового шифрування в Україні, вирішення цієї задачі є дуже актуальним і для нашої держави.

Проведений аналіз доступних наукових публікацій показує, що на сьогодні найбільш потужний клас атак на синхронні потокові шифри складають атаки, які будуються на основі наближень булевих функцій, пов’язаних з алгоритмами шифрування, функціями менш складної структури. Не дивлячись на помітний прогрес у розробці таких атак, низка важливих наукових задач залишається невирішеною. Зокрема, на сьогодні відсутні методи, які б дозволяли за достатньо загальних умов обґрунтовувати стійкість синхронних потокових шифрів відносно зазначених атак. Це свідчить про певне протиріччя між потребами в обґрунтовано стійких та практичних алгоритмах потокового шифрування, що необхідні для забезпечення безпеки державних інформаційних ресурсів, з одного боку, та відсутністю методів обґрунтування стійкості цих

алгоритмів відносно широкого кола сучасних атак, з іншого.

В дисертаційній роботі отримане нове вирішення **актуальної наукової задачі**, яка полягає в розробці методу побудови науково обґрунтованих оцінок стійкості потокових шифрів відносно статистичних атак, що базуються на алгебраїчно вироджених наближеннях булевих функцій.

Для вирішення поставленої наукової задачі **використано методи** теорії булевих функцій, гармонічного аналізу, лінійної алгебри, теорії ймовірностей та математичної статистики. Експериментальні дослідження та чисельні розрахунки виконувалися за допомогою платформи .NET Framework 4 (мови програмування – C#), а також пакету прикладних програм Maple 15 на персональному комп'ютері з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам'яті 4 Гб RAM на базі 32-розрядної ОС Windows 7 Service Pack 1.

Основні наукові та практичні результати, отримані в дисертації.

1. Запропоновано статистичні атаки на синхронні потові шифри, які узагальнюють низку раніше відомих, зокрема, атаку FKM [10] та кубічну атаку [9], і базуються на алгебраїчно вироджених наближеннях булевих функцій. Отримані аналітичні оцінки трудомісткості запропонованих атак свідчать про їх більш високу ефективність у порівнянні з аналогічними раніше відомими. Зокрема, застосування однієї з цих атак до редукованої версії шифру Grain-128 дозволяє зменшити обчислювальну складність відновлення ключа шифру майже в 2^{27} разів у порівнянні з атакою FKM.

2. Запропоновано метод побудови списку усіх високоймовірних k-вимірних наближень булевих функцій, який суттєво покращує раніше відомий метод [24] (має меншу трудомісткість в порівнянні з останнім, у певних випадках – в 1000 та більше разів). Зменшення трудомісткості запропонованого методу в порівнянні з методом [24] досягається за рахунок застосування більш точної оцінки кількості шуканих наближень вхідної функції

(твердження 3.1), а також більш економної організації обчислень, яка базується на детальному аналізі структури цих наближень (твердження 3.3 – 3.5).

Результати практичного застосування запропонованого методу показують, що він дозволяє швидко знаходити високоймовірні наближення функцій від невеликої кількості змінних (наприклад, для функції Карле-Фенга [159] від $n = 5$ змінних час виконання алгоритму 3.4 при $k \le 4$ складає декілька секунд, див. табл. 3.4).

3. Вперше запропоновано метод обчислення значень нижніх меж відносної відстані між зрівноваженою булевою функцією та множиною всіх k-вимірних функцій від n змінних. Сутність методу полягає у статистичному оцінюванні відносної відстані за допомогою спеціально розробленого ймовірнісного алгоритму, складність якого залежить лінійно від n та поліноміально від величин, обернених до точності та ймовірності помилки алгоритму. Показано, що при малих значеннях k запропонований метод може бути ефективно використаний на практиці для обґрунтування стійкості функцій ускладнення синхронних потокових шифрів відносно узагальненої статистичної атаки.

Зокрема, згідно з результатами практичного застосування запропонованого методу до функції ускладнення шифру Achterbahn-80 [161], відносна відстань між нею та множиною 2-вимірних функцій є не менше ніж 0,4369 з достовірністю не менше $0{,}75$, в той час як точне значення цієї відстані дорівнює 0,4375. При цьому для оцінювання відносної відстані за допомогою запропонованого методу потрібно майже в 5 разів менше часу в порівнянні з алгоритмом обчислення її точного значення.

4. Запропоновано метод пошуку алгебраїчно вироджених наближень булевих функцій, який відрізняється за сутністю від відомих [10, 23, 24, 29] та базується на отриманих аналітичних умовах, яким задовольняють шукані наближення. На відміну від [23, 24], запропонований метод не має передумовою виконання будь-яких обмежень стосовно відстані, на якій треба відшукати наближення, а на відміну від [10, 29] він дозволяє знаходити

наближення з більш широкого класу булевих функцій. Крім того, за певних умов запропонований метод надає можливість переконуватися у відсутності зазначених наближень, що дозволяє використовувати його для обґрунтування практичної стійкості синхронних потокових шифрів відносно узагальненої статистичної атаки.

Зокрема, застосування розробленого методу до шифру SNOW-2.0, який є прототипом майбутнього національного стандарту потокового шифрування, показує, що для низки функцій-оракулів від 256 змінних з достовірністю не менше ніж 0,99 не існує 20-вимірних наближень, які знаходяться від цих функцій на відносній відстані не більше ніж 0,2. (При цьому час роботи комп'ютерної програми складає приблизно 187 годин на комп'ютері з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам'яті 4 Гб RAM на базі 32-розрядної ОС Windows 7 Service Pack 1). Це свідчить про практичну стійкість SNOW-2.0 відносно узагальненої статистичної атаки.

Достовірність результатів дисертаційної роботи забезпечується адекватністю припущень, які лежать в основі проведених наукових досліджень, а також коректним застосуванням відомих математичних методів. Результати проведених обчислювальних експериментів узгоджуються з отриманими теоретичними висновками.

Значення наукових результатів дисертації для теорії полягає в тому, що вони утворюють наукову основу для вирішення задач оцінювання та обґрунтування стійкості перспективних синхронних потокових шифрів відносно найбільш потужних на сьогодні атак.

Практичне значення роботи полягає в тому, що отримані в ній результати дозволяють:

– розширити клас наближень булевих функцій, що можуть бути використані для побудови статистичних атак на синхронні потокові шифри;

– встановити науково обґрунтовані умови стійкості синхронних потокових шифрів відносно відомих та запропонованих статистичних атак;

– підвищити (у певних випадках – в 1000 та більше разів) трудомісткість найкращого з відомих алгоритмів побудови високоймовірних k-вимірних наближень булевих функцій;

– створити пакет прикладних програм для оцінювання та обґрунтування стійкості синхронних потокових шифрів відносно запропонованих статистичних атак;

– обґрунтувати практичну стійкість шифру SNOW 2.0, що є прототипом майбутнього національного стандарту потокового шифрування, відносно узагальненої статистичної атаки;

– підвищити обґрунтованість експертних висновків про застосування в Україні перспективних алгоритмів потокового шифрування, призначених для захисту державних інформаційних ресурсів.

Висновки та рекомендації по науковому та практичному використанні наукових результатів. Отримані в дисертаційній роботі нові наукові та практичні результати мають універсальний характер і можуть бути використані як при побудові, так і при дослідженні стійкості перспективних алгоритмів потокового шифрування.

1. Запропонована статистична атака на генератори гами з лінійним законом реініціалізації початкового стану (підрозділ 2.1) за певних умов дозволяє відновлювати ключі довжиною 128 біт зі складністю не більше 2^{38} елементарних операцій, використовуючи не більше 2^{15} відрізків гами довжиною не більше 117 знаків кожен разом з відповідними векторами ініціалізації.

2. Запропонована статистична атака на синхронні потокові шифри (підрозділ 2.2) узагальнює атаку FKM [10], кубічну атаку [9] і базується на наближеннях булевих функцій, що реалізуються алгоритмами шифрування, алгебраїчно виродженими функціями. Трудомісткість цієї атаки залежить

лінійно від складності обчислення значення функції-наближення та алгоритму опробування ключів, що використовується (див. формули (2.16), (2.17), (2.24) і теорему 2.2). Для випадку редукованої версії шифру Grain-128 трудомісткість узагальненої статистичної атаки не перевищує 2^{97}, що є в 2^{27} разів менше ніж трудомісткість раніше відомої атаки [10].

3. Запропонований ймовірнісний алгоритм, що використовується при проведенні узагальненої статистичної атаки (алгоритм 2.3), є модифікацією детермінованого алгоритму з [10], проте, на відміну від останнього, має теоретичне обґрунтування та дозволяє будувати "якісні" наближення з погляду відносної відстані до функції-оракула F (твердження 2.2). При цьому часова складність алгоритму дорівнює $O(T_F n^2)$, де T_F є складністю обчислення значення функції F, а n є кількістю її змінних.

4. Запропонований метод побудови списку всіх k-вимірних функцій степеня не вище d, що знаходяться на відносній відстані не більше $2^{-d}(1-\varepsilon)$ від булевої функції n змінних (підрозділ 3.1), суттєво покращує аналогічний раніше відомий метод [24] (має меншу трудомісткість в порівнянні з останнім, у певних випадках – в 1000 та більше разів). Зазначений метод може бути застосований на практиці при аналізі кореляційних властивостей функцій ускладнення синхронних потокових шифрів при малих значеннях k і d, якщо кількість "відносно великих" за модулем коефіцієнтів Уолша-Адамара функції f не перевищує 20.

5. Розроблені алгоритми 3.1 – 3.4 можуть бути використані для побудови k-вимірних наближень булевих функцій, що задаються за допомогою оракулів (а не тільки векторів значень). В цьому випадку на першому етапі алгоритмів 3.1 і 3.2 замість швидкого перетворення Адамара слід застосовувати один з відомих швидких алгоритмів побудови високоймовірних лінійних наближень вхідної функції, наприклад, вдосконалений алгоритм Левіна [134, 135].

6. Для обґрунтування відсутності високоймовірних k-вимірних наближень зрівноважених булевих функцій можна використовувати запропонований метод обчислення значень нижніх меж відносної відстані між заданою булевою функцією та множиною всіх k-вимірних функцій від n змінних (підрозділ 3.2). При малих значеннях k запропонований метод може бути ефективно використаний на практиці для обґрунтування стійкості функцій ускладнення синхронних потокових шифрів відносно статистичних атак, наведених в розділі 2. Зокрема, при $k \le 4$ цей метод дозволяє за декілька секунд побудувати список усіх високоймовірних k-вимірних наближень функції Карле-Фенга від $n = 5$ змінних.

7. Для знаходження (чи обґрунтування відсутності) певних алгебраїчно вироджених наближень булевих функцій, заданих за допомогою оракулів, можна використовувати метод, викладений в розділі 4. Метод складається з двох етапів, на першому з яких здійснюється пошук певних підпросторів, що є допустимими для вхідної функції-оракула (при цьому відсутність зазначених просторів свідчить про відсутність відповідних наближень). На другому етапі за знайденим підпростором будується булева функція від меншої кількості змінних, яка (поряд з базисом знайденого підпростору) задає наближення вхідної функції. Для розв'язання окремих задач на кожному етапі методу запропоновано поліноміальні ймовірнісні алгоритми, які можуть бути застосовані на практиці до функцій-оракулів від декількох десятків чи сотень змінних.

8. Застосування методу пошуку алгебраїчно вироджених наближень булевих функцій (розділ 4) до функції-оракула від 224 змінних, заданої редукованою версією шифру Grain-128, дозволяє за 53 години побудувати наближення цієї функції від 148 змінних, яке знаходиться від неї на відносній відстані $d \le 0{,}1702$. Отримане наближення надає можливість реалізувати узагальнену статистичну атаку на зазначену версію шифру, складність якої (за

умови теореми 2.1) не перевищує 2^{97}, в той час як складність раніше відомої атаки [10] дорівнює 2^{124}.

9. Застосування методу, викладеного у розділі 4, до низки функцій-оракулів від 256 змінних, заданих шифром SNOW-2.0, дає негативний результат. Зокрема, з достовірністю не менше ніж 0,99 жодна з зазначених функцій не має 20-вимірних наближень, які знаходяться від неї на відносній відстані не більше ніж 0,2 (час виконання відповідних комп'ютерних програм складає 185 – 187 годин). Цей факт свідчить про практичну стійкість шифру SNOW-2.0 відносно узагальненої статистичної атаки.

10. Основні наукові та практичні результати реалізовані в НДР "Севрюга", НДР "Мокрель" та науково-технічних розробках ЗАТ "Інститут інформаційних технологій". Вони також можуть бути використані при дослідженні стійкості перспективних потокових шифрів, призначених для захисту державних інформаційних ресурсів, зокрема, при розробці майбутнього національного стандарту потокового шифрування України. Подальший розвиток наукових ідей та методів, які лежать в основі дисертаційного дослідження, є актуальним напрямом в галузі забезпечення інформаційної безпеки держави.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Проект концепції інформаційної безпеки України. [Електронний ресурс] / Режим доступу: http://mip.gov.ua/done_img/d/30-project_08_06_15.pdf (дата звернення 07.11.2016) – Назва з екрану.

2) Про захист інформації в інформаційно-телекомунікаційних системах: Закон України від 05.07.1994 № 80/94-ВР // Відомості Верховної Ради України (ВВР). – 1994. – №31. – С. 286.

3) Фергюссон Н. Практическая криптография / Н. Фергюссон, Б. Шнайер. – М.: “Вильямс”, 2005. – 424 с.

4) Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. – М.: Триумф, 2002. – 816 с.

5) Ekdahl P. A new version of the stream cipher SNOW / P. Ekdahl, T. Johansson // Selected Areas in Cryptography – SAC 2002, – Springer-Verlag, 2003. – P. 47 – 61.

6) Hawkes P. Primitive Specification for SOBER-128 [Електронний ресурс] / P. Hawkes, G. Rose // Режим доступу: https://eprint.iacr.org/2003/081.pdf (дата звернення 07.11.2016). – Назва з екрану.

7) Daemen J. Fast Hashing and Stream Encryption with Panama [Електронний ресурс] / J. Daemen, C. Clapp // Режим доступу: http://mcrypt.hellug.gr/docs/panama.pdf.gz (дата звернення 07.11.2016). – Назва з екрану.

8) GSM Architecture and Interfaces [Електронний ресурс] / Режим доступу: https://www.pearsonhighered.com/samplechapter/0139491244.pdf (дата звернення 07.11.2016). – Назва з екрану.

9) Dinur I. Cube attacks on tweakable black box polynomials / I. Dinur, A. Shamir // Advances in Cryptology. – EUROCRYPT’09, Proceedings. – Springer-Verlag, 2009. – P.278 – 299.

10) Fischer S. Chosen IV statistical analysis for key recovery attacks on stream ciphers / S. Fischer, S. Khazaei, W. Meier // AFRICACRYPT 2008, Proceedings. – Springer-Verlag, 2008. – P. 236 – 245.

11) Aumasson J.-Ph. Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128 [Електронний ресурс] / J.-Ph. Aumasson, I. Dinur, L. Hensen, W. Meier, A. Shamir // Режим доступу: http://eprint.iacr.org/2009/218 (дата звернення 07.11.2016). – Назва з екрану.

12) Aumasson J.-Ph. Cube testers and key recovery attacks on reduced-raund MD6 and Trivium / J.-Ph. Aumasson, I. Dinur, W. Meier, A. Shamir // Fast Software Encryption. – FSE'09, Proceedings. – Springer-Verlag, 2009. – P. 1 – 22.

13) Aumasson J.-Ph. New features of latin dances: analysis of Salsa, ChaCha, and Rumba / J.-Ph. Aumasson, S. Fischer, S. Khazaei, W. Meier, C. Rechberger // Fast Softvare Encryption. – FSE 2008, Proceedings. – Springer-Verlag, 2008. – P. 470 – 488.

14) Dinur I. An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware [Електронний ресурс] / I. Dinur, T. Gueysu, C. Paar, A. Shamir, R. Zimmermann // Режим доступу: http://eprint.iacr.org/2011/282 (дата звернення 07.11.2016). – Назва з екрану.

15) Dinur I. Breaking Grain-128 with dynamic cube attacks / I. Dinur, A. Shamir // Fast Software Encryption – FSE'11, Proceedings. – Springer-Verlag. – 2011. – P. 167 – 187.

16) Faisal Sh. Extended cubes: enhancing cube attacks by low-degree non-linear equations / Sh. Faisal, M. Resa, W. Susilo, J. Seberry // Proc. of the 6-th ACM Symp. on Information, Comput. and Communication Security (ASIACCS'11), (Hong Kong, China, 22-24 March, 2011) – ACM, 2011. – P. 296 – 305.

17) Goldreich O. A hard core predicate for all one-way functions / O. Goldreich, L.A. Levin // Proc. 21 ACM Sympos. of Theory of Computing (STOC'89), (New York, USA, 1989). – ACM, 1989. – P. 25 – 32.

18) Goldreich O. Learning polynomials with queries: the highly noisy case / O. Goldreich, R. Rubinfeld, M. Sudan // SIAM J. Discrete Math. – Vol. 13. – № 4. – 2000. – P. 535 – 570.

19) Gopalan P. List decoding Reed-Muller codes over small fields / P. Gopalan, A. Klivans, D. Zuckerman // Proc. 40 ACM Symp. of Theory of Computing (STOC'08), (Victoria, British Columbia, Canada, 17-20 May, 2008). – ACM, 2008. – P. 265 – 274.

20) Kabatiasky G. List decoding of Reed-Muller codes / G. Kabatiasky, C. Tavernier // Proc. Ninth Int. Workshop on Algebraic and Comb. Coding Theory (ACCT'04). – 2004. – P. 230 – 235.

21) Trevisan L. Some applications of coding theory in computational complexity [Електронний ресурс] / L. Trevisan // Режим доступу: http://eccc.hpi-web.de/report/2004/043/download/ (дата звернення 07.11.2016). – Назва з екрану.

22) Fourquet R. Finding good linear approximations of block ciphers and its application to cryptanalysis of redused round DES [Електронний ресурс] / R. Fourquet, P. Loidreau, C. Tavernier // Режим доступу: http://ced.tavernier.free.fr/index_fichiers/Articles/Linear_Approximations.pdf (дата звернення 07.11.2016). – Назва з екрану.

23) Gopalan, P. Testing Fourier dimensionality and sparsity / P. Gopalan, R. O'Donnel, A. Servedio et all // SIAM J. on Computing. – V. 40(4). – 2011. – P. 1075–1100.

24) Gopalan, P. A Fourier-analytic approach to Reed-Muller decodind / P. Gopalan // 2013 54th IEEE Annual Symp. on Foudation in Computer Sci (FOCS'10). Proceedings, (Las Vegas, Nevada, USA, 23-26 October, 2010) – IEEE Computer Society, 2010. – P. 685 – 694.

25) Daemen J. Resynchronization weaknesses in synchronous stream ciphers / J. Daemen, R. Govaerts, J. Vandewalle // Advances in Cryptology. – EUROCRYPT'93, Proceedings. – Springer-Verlag. – 1993. – P. 159 – 167.

26) Borissov Y. On a resynchronization weakness in a class of combiners with memory / Y. Borissov, S. Nikova, B. Preneel, J. Vandewalle // The 3rd Conf. on Security in Communication Networks, Proceedings. – Springer-Verlag. – 2003. – P. 165 – 177.

27) Golić J. On the resynchronization attack / J. Golić, G. Morgari // Fast Software Encryption. – FSE'03, Proceedings. – Springer-Verlag. – 2003. – P. 100 – 110.

28) Armknecht F. Extending the resynchronization attack / F. Armknecht, J. Lano, B. Preneel // Selected Areas in Cryptography – SAC'04, Proceedings. – Springer-Verlag. – 2004. – P. 19 – 38.

29) Алексеев Е.К. О некоторых мерах нелинейности булевых функций / Е. К. Алексеев // Прикладная дискретная математика. – 2011. – № 2(12). – С. 5 – 16.

30) Алексейчук А.Н. Статистическая атака на генератор гаммы с линейным законом реинициализации начального состояния и функцией усложнения, близкой к алгебраически вырожденной / А.Н. Алексейчук, С.Н. Конюшок, А.Ю. Сторожук // Радиотехника. – 2014. – Вып. 176. – С. 13 – 21.

31) Олексійчук А.М. Швидкий імовірнісний алгоритм оцінювання відстані між зрівноваженою булевою функцією та множиною k-вимірних функцій / А.М. Олексійчук, С.М. Конюшок, А.Ю. Сторожук // Прикладная радиоэлектроника. – 2014. – Т.13. – №3. – С.186 – 191.

32) Сторожук А.Ю. Дослідження теоретичного обсягу матеріалу при моделюванні статистичної атаки на генератор гами з лінійним законом реініціалізації та функцією ускладнення, що є близькою до алгебраїчно виродженої. // А.Ю. Сторожук // Спеціальні телекомунікаційні системи та захист інформації. Збірник наукових праць. – 2014. – Вип. 2 (26). – С. 12 – 17.

33) Олексійчук А.Ю. Швидкі алгоритми побудови k-вимірних наближень наближень булевих функцій // А.М. Олексійчук, С.М. Конюшок, А.Ю. Сторожук // Захист інформації. – 2015. – Т. 17. – № 1. – С. 43 – 52.

34) Алексейчук А.Н. Обобщенная статистическая атака на синхронные поточные шифры // А.Н. Алексейчук, С.Н. Конюшок, А.Ю. Сторожук // Захист інформації. – 2015. – Т. 17. – № 3. – С. 54 – 65.

35) Олексійчук А.М. Метод пошуку алгебраїчно вироджених наближень булевих функцій для побудови статистичних атак на синхронні потокові шифри / А.М. Олексійчук, С.М. Конюшок, А.Ю. Сторожук // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні. – 2016. – Вип. 1 (31) 2016. – С. 65 – 79.

36) Конюшок С.Н. Криптографически безопасные генераторы псевдослучайных последовательностей, встроенные в операционные системы Windows и Linux / С. Н. Конюшок, А. Ю. Сторожук // Тези доповідей XV ювілейної міжнародної науково-практичної конференції [“Безопасность информации в информационно-телекоммуникационных системах”] (Київ, 22-25 травня 2012 р.). – К.: ООО “ИП ЭДЕЛЬВЕЙС”, НИЦ “ТЕЗИС” НТУУ “КПИ”, 2012. – С. 46.

37) Алексеев, Е. К. Об атаке на фильтрующий генератор с функцией усложнения, близкой к алгебраически вырожденной / Е. К. Алексеев // Сборник статей молодых ученых факультета МВК МГУ. – 2011. – В. 8. – С. 114 – 123.

38) Сторожук А.Ю. Модифицированная атака на фильтрующий генератор гаммы с линейным законом реинициализации и функцией усложнения близкой к алгебраически вырожденной / А.Ю. Сторожук, С.Н. Конюшок // Тези доповідей XVI міжнародної науково-практичної конференції [“Безопасность информации в информационно-телекоммуникационных системах”] (Київ, 21-24 травня 2013 р.). – К.: ООО “ИП ЭДЕЛЬВЕЙС”, НИЦ “ТЕЗИС” НТУУ “КПИ”, 2013. – С. 40 – 41.

39) Сторожук А.Ю. Статистическая атака на генератор гаммы с линейным законом реинициализации начального состояния и функции усложнения, близкой к алгебраически вырожденной / А.Ю. Сторожук, С.Н. Конюшок, С.Ж. Піскун // Праці міжнародної наукової конференції [“Питання оптимізації

обчислень (ПОО-XL)”] (АР Крим, Велика Ялта, смт. Куцівелі, 30 вересня – 4 жовтня 2013 р.). – К.: Інститут кібернетики імені В.М. Глушкова НАН України, 2013. – С. 256 – 257.

40) Алексейчук А.Н. Статистическая атака на генератор гаммы с линейным законом реинициализации начального состояния и функцией усложнения, близкой к алгебраически вырожденной / А.Н. Алексейчук, С.Н. Конюшок, А.Ю. Сторожук // Материалы междунар. науч. конгресса [“Международный конгресс по информатике: информационные системы и технологии”] (Білорусь, Мінськ, 4-7 листопада 2013 р.). – К.: Минск: БГУ, 2013. – С. 24 – 27.

41) Alekseychuk A.N. Algebraic degenerate approximations of Boolean functions for key recovery attacks on stream ciphers / A.N. Alekseychuk, S.N. Konushok, A.Y. Storozhuk // International conference “Probability, reliability and stochastic optimization”. Conference materials, (p. 27). Kyiv: Taras Shevchenko national university of Kyiv.

42) Конюшок С.М. Дослідження теоретичного значення обсягу матеріалу при моделюванні статистичної атаки на генератор гами з лінійним законом реініціалізації та функцією ускладнення, що є близькою до алгебраїчно виродженої / С.М. Конюшок, А.Ю. Сторожук // Тези доповідей XVII міжнародної науково-практичної конференції [“Безпека інформації в інформаційно-телекомунікаційних системах”] (Київ, 26-28 травня 2015 р.). – К.: Державна служба спеціального зв’язку та захисту інформації України, 2015. – С. 25 – 26.

43) Алексейчук А.Н. Усовершенствованный метод построения статистических атак на синхронные поточные шифры / А.Н. Алексейчук, С.Н. Конюшок, А.Ю. Сторожук. // Праці міжнародної наукової конференції [“Питання оптимізації обчислень (ПОО-XLII)”] (Закарпатська обл., Мукачівський район, смт. Чинадієво, 21-25 вересня 2015 р.). – К.: Інститут кібернетики імені В.М. Глушкова НАН України, 2015. – С. 136 – 137.

44) Олексійчук А.М. Метод пошуку алгебраїчно вироджених наближень булевих функцій для побудови статистичних атак на синхронні потокові шифри / А.М. Олексійчук, С.М. Конюшок, А.Ю. Сторожук. // Матеріали XVIII міжнародної науково-практичної конференції (випуск 18) [“Безпека інформації в інформаційно-телекомунікаційних системах”] (Київ, 25-26 травня 2016 р.). – К.: Державна служба спеціального зв’язку та захисту інформації України, 2015. – С. 38.

45) Про Державну службу спеціального зв’язку та захисту інформації України: Закон України від 23.02.2006 № 3475-IV // Відомості Верховної Ради України (ВВР). – 2006. – №30. – С. 258.

46) NESSIE: New European Schemes for Signatures, Integrity, and Encryption [Електронний ресурс] / Режим доступу: https://www.cosic.esat.kuleuven.be/nessie/ (дата звернення 07.11.2016). – Назва з екрану.

47) ECRYPT: The home page eSTREAM, the ECRYPT Stream Cipher Projeect [Електронний ресурс] / Режим доступу: http://www.ecrypt.eu.org/stream (дата звернення 07.11.2016). – Назва з екрану.

48) Biryukov A. Block Ciphers and Stream Ciphers: The State of the Art [Електронний ресурс] / A. Biryukov // Режим доступу: https://www.researchgate.net/publication/2895551_Block_Ciphers_and_Stream_Ciphers_The_State_of_the_Art (дата звернення 07.11.2016). – Назва з екрану.

49) Потий А. В. Исследование методов криптоанализа поточных шифров / А. В. Потий, Ю. А. Избенко // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні. – 2003. – Вип. 6. – С. 34 – 49.

50) Heine G. GSM Networks: Protocols, Terminology and Implementation [Електронний ресурс] / G. Heine // Режим доступу: http://ss7.at.ua/_ld/0/13_GSMM.pdf (дата звернення 08.11.2016). – Назва з екрану.

51) Specification of the Bluetooth System, Version 1.1 [Електронний ресурс] / Режим доступу: https://www.inf.ethz.ch/personal/hvogt/proj/btmp3/Datasheets/

Bluetooth_11_Specifications_Book.pdf (дата звернення 07.11.2016). – Назва з екрану.

52) Anderson R. Chameleon – A New Kind of Stream Cipher [Електронний ресурс] / Anderson R. // Режим доступу: https://www.cl.cam.ac.uk/~rja14/Papers/chameleon.pdf (дата звернення 08.11.2016). – Назва з екрану.

53) Akyildiz I.F. A Survey on Sensor Networks / I.F. Akyildiz, Su W., Y. Sankarasubramaniam, E. Cayirci // IEEE Communication Magazine. – Vol. 40, № 8, 2002. – P. 102 – 114.

54) Rodrigues F. The impact of Wireless Sensors in Buildings Automation [Електронний ресурс] / F. Rodrigues, C. Cardeira, J.M.F. Calado // Режим доступу: http://www.dem.ist.utl.pt/~cardeira/papers/Ibersensor007.pdf (дата звернення 08.11.2016). – Назва з екрану.

55) Chatterjee A. Practical applications of wireless sensor network based on military, environmental, health and home applications: a survey / A. Chatterjee, M. Pandey // International Journal of Scientific & Engineering Research. – Vol. 5, Issue 1, 2014. – P. 1043 – 1050.

56) Panayiotou C.G. Environmental monitoring using wireless sensor networks [Електронний ресурс] / C.G. Panayiotou, Fatta D., Michaelides M.P. // Режим доступу: http://www.eng.ucy.ac.cy/Christos/Papers/PFM_WMCCS.pdf (дата звернення 08.11.2016). – Назва з екрану.

57) Barrenetxea G. Wireless sensor networks for environmental monitoring: the sensorscope experience [Електронний ресурс] / G. Barrenetxea, F. Ingelrest, G. Schaefer, M. Vetterli // Режим доступу: https://gsfr.github.io/pdf/izs2008.pdf (дата звернення 08.11.2016). – Назва з екрану.

58) Ko J. Wireless sensor networks for healthcare [Електронний ресурс] / J. Ko, C. Lu, M.B. Srivastava, J. Stankovic, A. Terzis, M. Welsh // Режим доступу: https://www.cs.virginia.edu/~stankovic/psfiles/proceedings09-2.pdf (дата звернення 08.11.2016). – Назва з екрану.

59) Mohammadi S. A comparison of physical attacks on wireless sensor networks / S. Mohammadi H. Jadidoleslamy // International Journal of Peer to Peer Networks (IJP2P). – Vol.2, No 2, 2011. – P. 24 – 42.

60) Karlof C. Secure routing in wireless sensor networks: Attacks and countermeasures / C. Karlof, D. Wagner // AdHoc Networks Journal. – Vol.1, 2003. – P. 293 – 315.

61) Krontiris I. Cooperative intrusion detection in wireless sensor networks / I. Krontiris, Benenson Z., Giannetsos T., Freiling F., Dimitriou T. // Embedded Wireless Systems and Networks. – EWSN'09. – Springer-Verlag, 2009. – P. 263 – 278.

62) Perrig A. Security in wireless sensor networks / A. Perrig, J. Stankovic, D. Wagner // Communication of the ACM. – Vol. 47, No. 6, 2004. – P. 53 – 57.

63) Islam S. A hierarchical intrusion detection system in wireless sensor networks / S. Islam, R.H. Khan, D.M. Bappy // International Journal of Computer Science and Network Security. – Vol.10, No. 8, 2010. – P. 21 – 26.

64) Алферов А.П. Основы криптографии / А.П. Алферов, А.Ю. Зубов, А.С. Кузьмин, А.В. Черемушкин. – М.: Гелиос АРВ, 2001. – 480 с.

65) Driessen B. Eavesdropping on satellite telecommunication systems [Електронний ресурс] / B. Driessen // Режим доступу: https://pdfs.semanticscholar.org/0028/477ebfe6b3c636dd1f87a9cf1fc5e39cca9a.pdf (дата звернення 08.11.2016). – Назва з екрану.

66) Driessen B. Don't trust satellite phones: a security analysis of two satphone standards / B. Driessen, R. Hund, C. Willems, C. Paar, T. Holz // IEEE Symposium on Security and Privacy. – 2012. – P. 128 – 142.

67) Mousa A. Evaluation of the RC4 algorithm for data encryption / A. Mousa, A. Hamad // International Journal of Computer Science & Applications. – Vol. 3, No. 2, 2006. – P. 44 – 56.

68) Hell M. Security evaluation of stream cipher Enocoro-128v2 [Електронний ресурс] / M. Hell, T. Johanson // Режим доступу:

http://www.cryptrec.go.jp/estimation/techrep_id2008.pdf (дата звернення 08.11.2016). – Назва з екрану.

69) De Cannière C. Trivium specifications [Електронний ресурс] / C. De Cannière, B. Preneel // Режим доступу: http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf (дата звернення 08.11.2016). – Назва з екрану.

70) Watanabe D. MUGI pseudorandom number generator [Електронний ресурс] / D. Watanabe, S. Furuya, H. Yoshida, K. Takaragi // Режим доступу: http://www.sdl.hitachi.co.jp/crypto/mugi/index-e.htm (дата звернення 08.11.2016) – Назва з екрану.

71) Boesgaard M. The stream cipher Rabbit [Електронний ресурс]/ M. Boesgaard, M. Vesterager, T. Christensen // Режим доступу: http://blog.zhenglei.net/wp-content/uploads/2016/07/rabbit.pdf (дата звернення 08.11.2016). – Назва з екрану.

72) Berbain C. Decim^{v2} / C. Berbain, O. Billet, A. Canteaut, N. Courtois, B. Debraize, H. Gilbert, L. Goubin, A. Gouget, L. Granboulan, C. Lauradoux, M. Minier, T. Pornin, H. Sibert // Lecture Notes in Computer Science. – LNCS'08. – Springer-Verlag. – 2008. – P. 140 – 148.

73) Stream cipher KCipher-2 [Електронний ресурс]. / KDDI Corporation // Режим доступу: http://www.cryptrec.go.jp/english/cryptrec_13_spec_cypherlist_files/PDF/21_00espec.pdf (дата звернення 08.11.2016). – Назва з екрану.

74) Фомичев В.М. Дискретная математика и криптология / В.М. Фомичев. – М.: ДИАЛОГ-МИФИ, 2003. – 400 с.

75) Meneses A., van Oorschot P., Vanderstone S. Handbook of applied cryptography / A. Meneses, P. van Oorschot, S. Vanderstone. – USA, FL: CRC Press, 1996. – 816 P.

76) Бабаш А.В. Криптография / А.В. Бабаш, Г.П. Шанкин. – М.: Солон-Р, 2002. – 511с.

77) Ongoing research areas in symmetric cryptography [Електронний ресурс]. / ECRYPT summary report // Режим доступу: http://www.ecrypt.eu.org/documents/D.STVL.5-1.1 (дата звернення 08.11.2016). – Назва з екрану.

78) Zenner E. Stream cipher ctiteria [Електронний ресурс] / E. Zenner // Режим доступу: http://www.erikzenner.name/docs/2006_estream_criteria.pdf (дата звернення 08.11.2016). – Назва з екрану.

79) Словарь криптографических терминов / под ред. Б. А. Погорелова и В. Н. Сачкова. – М.: МЦНМО, 2006, – 94 с.

80) Hawkes P. On the applicability of distinguishing attacks against stream ciphers / P. Hawkes, G. Rose // The 3rd NESSIE Workshop, Proceedings. – 2002. – P. 6.

81) Coppersmith D. Cryptanalysis of stream ciphers with linear masking / D. Coppersmith, S. Halevi, C.S. Jutla // Advances in Cryptology. – CRYPTO’02, Proceedings. – Springer-Verlag. – 2002. – P. 515 – 532.

82) Tsunoo Y. Distinguishing attack against TPypy / Y. Tsunoo, Saito T., T. Kawabata, H. Nakashima // Selected Areas in Cryptography – SAC’07. – Springer-Verlag. – 2007. – P. 396 – 407.

83) Meier W. Fast correlation attacks: methods and coutermeasures // Lecture Notes in Computer Science. – FSE’2011, Proceedings. – Springer Verlag. – 2011. – P. 55 – 67.

84) Canteaut A. Fast correlation attacks against stream ciphers and related open problems // The 2005 IEEE Information Theory Workshop on Theory and Practice in Information-Theoretic Security (ITW’05), (Awaji Island, Japan, 16-19 October, 2005) – IEEE, 2005. – P. 49 – 54.

85) Biham E. New types of cryptoanalytic attacks using related keys // Advances in Cryptology. – EUROCRYPT’93, Proceedings. – Springer-Verlag. – 1993. – P. 340 – 357.

86) Логачев О.А. Булевы функции в теории кодирования и криптологии / О.А. Логачев, А.А. Сальников, В.В. Ященко. – М.: МЦНМО, 2004. – 470 с.

87) Шеннон К. Теория связи в секретных системах / К. Шеннон. Работы по теории информации и кибернетике; пер. с англ.; под. ред. Р.Л. Добрушина и О.Б. Лупанова. – М.: Изд-во иностр. литературы, 1963. – С. 333 – 402.

88) Golic J. Cryptanalysis of alleged A5 stream cipher // Advances in Cryptology – EUROCRYPT'97, Proceedings. – Springer-Verlag. – 1997. – P. 239 – 255.

89) Hellman M.E. A cryptographic time-memoty trade-off // IEEE Transactions on Information Theory. – Vol. 26, № 4. – 1980. – P. 401 – 406.

90) Babbage S. H. Improved exhaustive search attacks on stream ciphers // European Convention on Security and Detection, IEEE Conference publication. – 1995. – P. 161 – 166.

91) Biryukov A. Real time cryptanalysis of A5/1 on a PC / A. Biryukov, A. Shamir, D. Wagner // Fast Fast Software Encryption. – FSE'03, Proceedings. – Springer Verlag. – 2000. – P. 1 – 18.

92) Hong J. Rediscovery of time memory tradeoffs [Електронний ресурс] / J. Hong, P. Sarkar // Режим доступу: http://eprint.iacr.org/2005/090.pdf (дата звернення 08.11.2016). – Назва з екрану.

93) De Cannire C. Comments on the Rediscovery of Time Memory Tradeoffs [Електронний доступ] / C. De Cannire, L. Lano, B. Preneel // Режим доступу: http://www.ecrypt.eu.org/stream/papersdir/040.pdf (дата звернення 08.11.2016). – Назва з екрану.

94) Courtois N. Algebraic attacks on stream ciphers with linear feedback / N. Courtois, W. Meier // Advances in Cryptology. – EUROCRYPT '03, Proceedings. – Springer-Verlag. – 2003. – P. 345 – 359.

95) Algebraic cryptanalysis of symmetric primitives [Електронний ресурс]. / ECRYPT summary report // Режим доступу: https://hal.archives-ouvertes.fr/hal-00328626/file/D-STVL-7.pdf (дата звернення 08.11.2016). – Назва з екрану.

96) Courtois N. Higher order correlation attacks, XL algorithm and cryptanalysis of Toyocrypt / N. Courtois // Режим доступу: http://eprint.iacr.org/2002/087 (дата звернення 08.11.2016). – Назва з екрану.

97) Courtois N. Fast algebraic attacks on stream ciphers with linear feedback // Advances in Cryptologyю – CRYPTO'03, Proceedings. – Springer-Verlag. – 2003. – P. 177 – 194.

98) Armknecht F. A linearization attack on the bluetooth key stream generator [Електронний ресурс] / F. Armknecht // Режим доступу: http://eprint.iacr.org/2002/191 (дата звернення 08.11.2016). – Назва з екрану.

99) Armknecht F. Algebraic attacks on combiners with memory / F. Armknecht, M. Krause // Advances in Cryptology. – CRYPTO'03, Proceedings. – Springer-Verlag. – 2003. – P. 162 – 175.

100) Таранников Ю.В. Алгебраические атаки на потоковые шифры и алгебраическая иммунность булевых функций // Математика и безопасность информационных технологий. Материалы конференции в МГУ 2 – 3 ноября 2005 г. – М.: МЦНМО, 2006. – 480 с.

101) Zeng K. On the linear consistency test (LCT) in cryptanalysis with applications / K. Zeng, C. Yang, Y. Rao // Advances in Cryptology. – CRYPTO'89, Proceedings. – Springer-Verlag. – 1990. – P. 164 – 174.

102) Hawkes P. Rewriting variables: the complexity of fast algebraic attacks on stream ciphers / P. Hawkes, G. Rose // Advances in Cryptology. – CRYPTO'04, Proceedings. – Springer-Verlag. – 2004. – P. 390 – 406.

103) Meier W. Algebraic attacks and decomposition of Boolean functions / W. Meier, E. Pasalic, C. Carlet // Advances in Cryptology. – EUROCYPT'04, Proceedings. – Springer-Verlag. – 2004. – P. 474 – 491.

104) Armknecht F. Improving fast algebraic attacks / F. Armknecht // Fast Software Encryption. – FSE'04, Proceedings. – Springer-Verlag. – 2004. – P.65 – 82.

105) Armknecht F. On the existence of low-degree equatios for algebraic attacks [Електронний ресурс] / F. Armknecht // Режим доступу: http://eprint.iacr.org/2004/185 (дата звернення 08.11.2016). – Назва з екрану.

106) Joux A. Algorithmic cryptanalysis / A. Joux. – USA, FL: CRC Press, 2009. – 496 p.

107) Bard G.V. Algebraic cryptanalysis / G.V. Bard. – USA, NY: Springer, 2009. – 356 p.

108) Siegenthaler T. Decrypting a class of stream ciphers using ciphertext only / T. Siegenthaler // IEEE Transactions on Information Theory. – Vol. 34, № 1. – 1985. – P. 81 – 85.

109) Meier W. Fast correlation attacks on certain stream ciphers / W. Meier, O. Staffelbach // Journal of Cryptology. – Vol. 1, № 3. – 1989. – P. 159 – 167.

110) Jönsson F. Some results on fast correlation attacks [Електронний ресурс] / F. Jönsson // Режим доступу: http://www.pcc.lth.se/PrimePub/primepub.asp@AemneID=572&SpraakID=1&RotID=1.html (дата звернення 08.11.2016). – Назва з екрану.

111) Clark A. A comparison of fast correlation attacks / A. Clark, J. Golic, E. Dawson // Fast Software Encryption. – FSE’96, Proceedings. – Springer-Verlag. – 1996. – P. 145 – 158.

112) Chepyzhov V. A simple algorithm for fast correlation attacks on stream ciphers / V. Chepyzhov, T. Johansson, B. Smeets // Fast Software Encryption. – FSE’00, Proceedings. – Springer. – 2000. – P. 181 – 195.

113) Johansson T. Fast correlation attacks through reconstruction of linear polynomials / T. Johansson, F. Jonsson // Advances in Cryptology – CRYPTO’00, Proceedings. – Springer-Verlag. – 2000. – P. 300 – 315.

114) Mihaljevic M. A fast iterative algorithm for a shift register initial state reconstruction given the noisy output sequence / M. Mihaljevic, J. Golic // Advances in Cryptology – AUSCRYPT’90, Proceedings. – Springer-Verlag. – 1990. – P. 165 – 175.

115) Chepyzhov V. On a fast correlation attack on certain stream ciphers / V. Chepyzhov, B. Smeets // Advances in Cryptology. – EUROCYPT’91, Proceedings. – Springer-Verlag. – 1991. – P. 176 – 185.

116) Penzhorn W. Correlation attacks on stream ciphers: Computing low-weight parity checks based on error-correcting codes / W. Penzhorn // Fast Software Encryption. – FSE’96, Proceedings. – Springer. – 1996. – P. 159 – 172.

117) Saarinen M.-J. O. Chosen-IV Statistical Attacks on eSTREAM Stream Ciphers [Електронний ресурс] / M.-J. O. Saarinen // Режим доступу: http://www.ecrypt.eu.org/stream/papersdir/2006/013.pdf (дата звернення 08.11.2016). – Назва з екрану.

118) Turan M. Statistical Analysis of Synchronous Stream Ciphers [Електронний ресурс] / M. Turan, A. Docganaksoy, C. Calik // Режим доступу: http://www.ecrypt.eu.org/stream/papersdir/2006/012.pdf (дата звернення 08.11.2016). – Назва з екрану.

119) Mihaljevic M. A low-complexity and high-performance algorithm for the fast correlation attack / M. Mihaljevic, M. Fossorier, H. Imai // Fast Software Encryption. – FSE’00, Proceedings. – Springer-Verlag. – 2000. – P. 196 – 212.

120) Johansson T. Reduced complexity correlation attacks on two clock controlled generators / T. Johansson // Advances in Cryptology. – ASIACRYPT’98, Proceedings. – Springer-Verlag. – 1998. – P.342 – 356.

121) Canteaut A. Improved fast correlation attacks using parity-check equations of weight 4 and 5 / A. Canteaut, M. Trabbia // Advances in Cryptology. – EUROCYPT’00, Proceedings. – Springer-Verlag. – 2000. – P. 573 – 588.

122) Johansson T. Improved fast correlation attack on stream ciphers via convolutional codes / T. Johansson, F. Jonsson // Advances in Cryptology. – EUROCYPT’99, Proceedings. – Springer-Verlag. – 1999. – P. 347 – 362.

123) Johansson T. Fast correlation attacks based on Turbo Code techniques / T. Johansson, F. Jonsson // Advances in Cryptology. – CRYPTO’99, Proceedings. – Springer-Verlag. – 1999. – P. 181 – 197.

124) Johansson T. Theoretical analysis of a correlation attack based on convolutional codes / T. Johansson, F. Jonsson // IEEE Transactions on Information Theory. – Vol. 48, № 8. – 2002. – P. 2173 – 2181.

125) Johannesson R. Fundamentals of convolutional coding / R. Johannesson, K. Zigangirov. – USA, NY: IEEE Press, 1999. – 428 P.

126) Blondeau C. Accurate estimates on the data complexity and success probability for various cryptanalyses / C. Blondeau, B. Ge′rard, J.-P. Tillich // Design, Codes and Cryptography. – Vol. 59(1-3). – 2011. – P. 3 – 34.

127) Zhang B. Fast correlation attacks over extension fields, large-unit linear approximation and cryptanalysis of SNOW 2.0 / B. Zhang, C. Xu, W. Meier // Lecture Notes in Computer Science. – CRYPTO'15. – Springer-Verlag. – 2015. – P. 643 – 662.

128) Lohlein B. Attacks based on conditional correlations against the nonlinear filter generator [Електронний доступ] / B. Lohlein // Режим доступу: http://eprint.iacr.org/2003/020.pdf (дата звернення 08.11.2016). – Назва з екрану.

129) Lee S. Conditional correlation attack on non linear filter generators / S. Lee, S. Chee, S. Park // Advances in Cryptology. – ASIACRYPT'96, Proceedings. – Springer-Verlag. – 1996. – P. 360 – 367.

130) Golic J. Correlation properties of a general binary combiner with memory / J. Golic // Journal of Cryptology. – Vol. 9, № 2, – 1996. – P. 111 – 126.

131) Lechner, R. L. Harmonic analysis of switching functions / R. L. Lechner // Recent Developments in Switching Theory. – 1971. P. 122–228.

132) Dawson E. Construction of correlation immune Boolean functions / E. Dawson, C.K. Wu // Information and Communication Security, Proceedings. – Springer-Verlag. – 1997. – P. 170 – 180.

133) Думер И.И. Списочное декодирование двоичных кодов Рида-Маллера первого порядка / И.И. Думер, Г.А. Кабатянский, С. Тавернье // Проблемы передачи информации. – 2007. – Т. 43. – Вып. 3. – С. 66 – 74.

134) Levin L.A. Randomness and non-determinism / L.A. Levin // European Summer Meeting of the Association for Symbolic Logic. – J. Symbolic Logic, 1993. – V. 58. – No. 3. – P. 1102 – 1103.

135) Bshouty N. More efficient PAC-learning of DNF with membership queries under the uniform distribution / N. Bshouty, J. Jackson, C. Tamon // Proc. 12th Annual Conf. on Comput. Learning Theory. – ACM, 1999. – P. 286 – 295.

136) Abdouli A.S. The Goldreich-Levin algorithm with reduced complexity / A.S. Abdouli, I. Dumer, G. Kabatiansky, C. Tavernier // Thirteenth International Workshop on Algebraic and Combinatorial Coding Theory (ACCT'12), (Pomorie, Bulgaria, 15-21 June, 2012) – 2012. – P. 7 – 14.

137) Алексейчук А.Н. О статистических свойствах нелинейности сужений булевых функций на случайно выбранное подпространство / А.Н. Алексейчук, С.Н. Конюшок // Прикладная дискретная математика. – Вып. №1(15). – 2012. – С. 5 – 10.

138) Олексійчук А.М. Швидкі алгоритми статистичного оцінювання максимальних значень ймовірностей лінійних апроксимацій булевих функцій / А.М. Олексійчук, А.С. Шевцов // Спеціальні телекомунікаційні системи та захист інформації. – 2010. – Вип. 1(17). – С. 85 – 95.

139) Алексейчук А.Н. Быстрый алгоритм статистического оценивания максимальной несбалансированности билинейных аппроксимаций булевых отображений / А.Н. Алексейчук, А.С. Шевцов // Прикладная дискретная математика. – Вып. №3(13). – 2011. – С.5 – 11.

140) Canteaut, A. On the correlations between a combining function and function of fewer variables / A. Canteaut // The 2002 IEEE Information Theory Workshop, Proceedings (ITW'02), (Bangalore, India, 20-25 October, 2002) – Springer-Verlag, 2002. – P.78 – 81.

141) Алексейчук, А. Н. Усовершенствованный тест k-мерности для булевых функций / А.Н.Алексейчук, С. Н. Конюшок // Кибернетика и системный анализ. – Т. 49, № 2. – 2013. – С. 27 – 35.

142) Alekseychuk, A. N. Fast algorithm for reconstruction of high-probable low-dimensional approximations for Boolean functions / A. N. Alekseychuk, S. N. Konyushok // Modern Stochastics: Theory and Applications III, Proceedings. Kyiv. Taras Shevchenko National University. – 2012. – P. 32.

143) Алексейчук А.Н. Алгебраически вырожденные приближения булевых функций / А.Н. Алексейчук, С.Н. Конюшок // Кибернетика и системный анализ. – Т. 50, №6. – 2014. – С. 3 – 14.

144) Aumasson J.-Ph. New features of latin dances: analysis of Salsa, ChaCha, and Rumba / J.-Ph. Aumasson, S. Fischer, S. Khazaei, W. Meier, C. Rechberger // Fast Softvare Encryption. – FSE 2008, Proceedings. – Springer-Verlag. – 2008. – P. 470 – 488.

145) Алексейчук А.Н. Об эффективности метода вероятностно нейтральных битов в статистическом криптоанализе синхронных поточных шифров / А.Н. Алексейчук, С.Н. Конюшок // Кибернетика и системный анализ. – Т. 52, №4. – 2016. – С. 3 – 10.

146) Бойко Ю.М. Концептуальні особливості реалізації безпровідних сенсорних мереж / Ю.М. Бойко, В.М. Локазюк, В.В. Мішан // Вісник Хмельницького національного університету. – 2010. – № 2. – С. 94 – 97.

147) Pottie G.J. Wireless integrated network sensors / G.J. Pottie, W.J. Kaiser // Commun. ACM, 43(5). – 2000. – P. 51– 58.

148) Mainwaring A. Wireless sensor networks for habital monitoring / A. Mainwaring, J.Polastre, R. Szewczyk, D.Culler, J. Anderson // WSNA’02, Proceedings. – 2002. – P. 88 – 97.

149) Tateson J. Real world issues in deploying a wireless sensor network for oceanography [Електронний ресурс] / J. Tateson, C. Roadknight, A. Gonzalez, T. Khan, S. Fitz, Ian Henning, N. Boyd, C. Vincent, I. Marshall // Режим доступу:

http://www.lancaster.ac.uk/staff/marshai2/papers/realwsn05.pdf (дата звернення 08.11.2016). – Назва з екрану.

150) Milenkovic A. Wireless sensor networks for personal health monitoring: issues and an implementation [Електронний ресурс] / A. Milenkovic, C. Otto, E. Jovanov // Режим доступу: http://www.ece.tufts.edu/ee/194HHW/papers/milenkovic_compcomm06.pdf (дата звернення 08.11.2016). – Назва з екрану.

151) Faludi R. Building wireless sensor networks [Електронний ресурс] / Режим доступу: http://ab-log.ru/files/File/books/WirelessSensorNetwork.pdf (дата звернення 08.11.2016). – Назва з екрану.

152) Hill J.L. System architecture for wireless sensor networks [Електронний ресурс] / J.L. Hill // Режим доступу: http://eps2009.dj-inod.com/docs/09-02-01/system_architecture_for_wireless_sensor_networks.pdf (дата звернення 08.11.2016). – Назва з екрану.

153) Motolla L. Programming wireless sensor networks: fundamental concepts and state of the art [Електронний ресурс] / L. Motolla // Режим доступу: https://www.sics.se/~luca/papers/mottola10programming.pdf (дата звернення 08.11. 2016). – Назва з екрану.

154) Liang Y. Routing topology interface for wireless sensor networks [Електронний ресурс] / Y. Liang // Режим доступу: http://www.sigcomm.org/sites/default/files/ccr/papers/2013/April/2479957-2479961.pdf (дата звернення 08.11.2016). – Назва з екрану.

155) Patwari N. Relative location estimation in wireless sensor networks [Електронний ресурс] / N. Patwari // Режим доступу: http://web.eecs.umich.edu/~hero/Preprints/patwari_tsp02_final.pdf (дата звернення 08.11.2016). – Назва з екрану.

156) Ширяев А.Н. Вероятность: Учеб. пособие для вузов / А.Н. Ширяев. – М.: Наука, 1989. – 640 с.

157) De Wolf R. A brief introduction to Fourier analysis on the Boolean cube / R. De Wolf // Theory of Comput. Library. – No. 1. – 2008. – P. 1 – 20.

158) Сачков В.Н. Введение в комбинаторные методы дискретной математики / В.Н. Сачков. – М.: МНЦНМО, 2004. – 424 с.

159) Carlet C. An infinite class of balanced functions with optimal algebraic immunity, good immunity to fast algebraic attacks, and good nonlinearity / C. Carlet, K. Feng // Advances in Cryptology. – ASIACRYPT'08, Proceedings. – Springer-Verlag. – 2008. – P. 425 – 440.

160) Глухов М.М. Алгебра / М.М. Глухов, В.П. Елизаров, А.А. Нечаев. – Учебник в 2-х т., Т. 1. – М.: Гелиос АРВ, 2003. – 336 с.

161) Gammel B.M. Achterbahn-128/80 [Електронний ресурс] / B.M. Gammel, R. Gottfert, O. Kniffler // Режим доступу: http://www.matpack.de/achterbahn/Gammel_Goettfert_Kniffler_Achterbahn-128-80.pdf (дата звернення 10.11.2016). – Назва з екрану.

162) Gowers T. A new proof of Szemere'di's theorem / T. Gowerts // Geom. Funct. Anal. – Vol. 11(3). – 2001. – P. 465 – 588.

163) Hoeffding W. Probability inequalities for sums of bounded random variables / W. Hoeffding // J. Amer. Statist. Assoc. – Vol. 58. – № 301. – 1963. – P. 13 – 30.

ДОДАТКИ

Додаток А

Програмний код реалізації алгоритму відновлення ключа генератора гамии з лінійним законом реініціалізації початкового стану та функцією ускладнення, що є близькою до алгебраїчно виродженої

Програмна реалізація алгоритму відновлення ключа генератора гамии з лінійним законом реініціалізації початкового стану та функцією ускладнення, що є близькою до алгебраїчно виродженої виконана на ПК з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам'яті 4 Гб RAM на базі 32-розрядної ОС Windows 7 Service Pack 1. Мова програмування – C# .NET Framework 4.0. Середовище розробки – Microsoft Visual Studio 2010.

```
//Файл Matrix.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using Org.BouncyCastle.Math;

namespace Checking
{
    static class Matrix
    {

        public static bool[,] A;
        public static bool[,] B;
        public static bool[,] M;
        public static bool[,] L;
        public static bool[,] P;

        private static string DIR = Directory.GetCurrentDirectory() +
"/Example_1/random.txt";

        public static void Set(int N, int l0, int l1, int s, int n, string polinom, string
variables)
        {
            A = new bool[N, l0];
            MatrixFromFile(A, Directory.GetCurrentDirectory() +
"/Example_1/A_matrix_128x128.txt");

            B = new bool[N, l1];
```

```
            MatrixFromFile(B, Directory.GetCurrentDirectory() +
"/Example_1/B_matrix_128x128.txt");

            M = new bool[s, n];
            MatrixFromFile(M, Directory.GetCurrentDirectory() +
"/Example_1/M_matrix_2x64.txt");

            L = GenerateL(polinom);
            P = GenerateP(N, variables);
        }


        private static bool[,] GenerateA(int N, int l0)
        {
            StreamReader sr = new StreamReader(DIR);
            Random r = new Random();
            string BIGSTRING = sr.ReadToEnd();
            string[] kk = new string[1024];
            for (int i = 0; i < kk.Length; i++)
            {
                kk[i] = BIGSTRING.Substring(r.Next(BIGSTRING.Length - 1), 1);
            }
            sr.Close();
            BigInteger n = new BigInteger(ASCIIEncoding.ASCII.GetBytes(string.Concat(kk)));
            string[] gamma = new string[N];
            for (int i = 0; i < N; i++)
            {
                gamma[i] = n.ToString(2).Substring(r.Next(n.ToString(2).Length - l0), l0);
            }
            bool[][] matrix = new bool[N][];
            for (int i = 0; i < N; i++)
            {
                matrix[i] = new bool[l0];
                for (int j = 0; j < l0; j++)
                {
                    if (int.Parse(gamma[i].Substring(j, 1)) == 1)
                    { matrix[i][j] = true; }
                    else
                    { matrix[i][j] = false; }
                }
            }
            return ConvertMatrix(matrix);
        }


        private static bool[,] GenerateM(int s, int n)
        {
            StreamReader sr = new StreamReader(DIR);
            Random r = new Random();
            string BIGSTRING = sr.ReadToEnd();
            string[] kk = new string[1024];
            for (int i = 0; i < kk.Length; i++)
            {
                kk[i] = BIGSTRING.Substring(r.Next(BIGSTRING.Length - 1), 1);
            }
            sr.Close();
            BigInteger N = new BigInteger(ASCIIEncoding.ASCII.GetBytes(string.Concat(kk)));
            string[] gamma = new string[s];
            for (int i = 0; i < s; i++)
            {
                gamma[i] = N.ToString(2).Substring(r.Next(N.ToString(2).Length - n), n);
            }
```

```
        bool[][] matrix = new bool[s][];
        for (int i = 0; i < s; i++)
        {
            matrix[i] = new bool[n];
            for (int j = 0; j < n; j++)
            {
                if (int.Parse(gamma[i].Substring(j, 1)) == 1)
                { matrix[i][j] = true; }
                else
                { matrix[i][j] = false; }
            }
        }
        return ConvertMatrix(matrix);
    }


    private static bool[,] GenerateB(int N, int l1)
    {
        StreamReader sr = new StreamReader(DIR);
        Random r = new Random();
        byte[] s = new byte[N];
        string BIGSTRING = sr.ReadToEnd();
        string[] kk = new string[1024];
        for (int i = 0; i < kk.Length; i++)
        {
            kk[i] = BIGSTRING.Substring(r.Next(BIGSTRING.Length - 1), 1);
        }
        sr.Close();
        BigInteger n = new BigInteger(ASCIIEncoding.ASCII.GetBytes(string.Concat(kk)));
        string[] gamma = new string[N];
        for (int i = 0; i < N; i++)
        {
            gamma[i] = n.ToString(2).Substring(r.Next(n.ToString(2).Length - l1), l1);
        }

        bool[][] matrix = new bool[N][];
        for (int i = 0; i < N; i++)
        {
            matrix[i] = new bool[l1];
            for (int j = 0; j < l1; j++)
            {
                if (int.Parse(gamma[i].Substring(j, 1)) == 1)
                { matrix[i][j] = true; }
                else
                { matrix[i][j] = false; }
            }
        }
        return ConvertMatrix(matrix);
    }


    private static bool[,] GenerateL(string polinom)
    {
        string f = polinom;
        char[] param = new char[] { ',' };
        string[] splitted = f.Split(param);
        int LFSR_lenght = int.Parse(splitted.Last());
        bool[][] result = new bool[LFSR_lenght - 1][];
        for (int i = 0; i < LFSR_lenght - 1; i++)
        {
            result[i] = new bool[LFSR_lenght];
            for (int j = 0; j < LFSR_lenght; j++)
```

```
            {
                result[i][j] = false;
                result[i][i + 1] = true;

            }
        }
        bool[] LastRow = new bool[int.Parse(splitted[splitted.Length - 1])];
        for (int i = 0; i < LastRow.Length; i++)
        {
            for (int j = 0; j < splitted.Length - 1; j++)
            {
                LastRow[i] = false;
                LastRow[int.Parse(splitted[j])] = true;
            }
        }
        List<bool[]> F = new List<bool[]>();
        for (int i = 0; i < LFSR_lenght - 1; i++)
        {
            F.Add(result[i]);
        }
        F.Add(LastRow);
        result = F.ToArray();

        return ConvertMatrix(result);
    }
    private static bool[,] GenerateP(int columns, string variables)
    {
        string f = variables;
        char[] param = new char[] { ',' };
        string[] splitted = f.Split(param);
        int[] FI = new int[columns];

        for (int i = 0; i < FI.Length; i++)
        {
            for (int j = 0; j < splitted.Length - 1; j++)
            {
                FI[i] = 0;
                FI[int.Parse(splitted[j])] = 1;
            }
        }
        bool[] l = new bool[FI.Length];
        bool[][] result = new bool[splitted.Length - 1][];

        for (int i = 0; i < FI.Length; i++)
        {
            if (FI[i] != 0)
            {
                l[i] = true;
            }
            else
            {
                l[i] = false;
            }
        }
        for (int k = 0; k < splitted.Length - 1; k++)
        {
            result[k] = new bool[columns];
            for (int i = 0; i < columns; i++)
            {
                if (l[i] == true)
                {
                    for (int g = 0; g < columns; g++)
```

```
                {
                    result[k][g] = false;
                    result[k][i] = true;
                }
                l[i] = false;
                break;
            }
        }
    }
    return ConvertMatrix(result);
}


private static bool[,] GenerateGOOD_B(int rows, int columns)
{
    bool[,] GoodB1 = GenerateB(rows, columns);
    bool[,] GoodB2 = new bool[rows, columns];
    int RANK = 0;
    if (rows >= columns)
    {
        while (RANK < columns)
        {
            GoodB1 = GenerateB(GoodB1.GetLength(0), GoodB1.GetLength(1));
            Array.Copy(GoodB1, GoodB2, rows * columns);
            RANK = Rank.Of(GoodB1);
        }
    }
    else
    {
        while (RANK < rows)
        {
            GoodB1 = GenerateB(GoodB1.GetLength(0), GoodB1.GetLength(1));
            Array.Copy(GoodB1, GoodB2, rows * columns);
            RANK = Rank.Of(GoodB1);
        }
    }
    return GoodB2;
}
private static bool[,] GenerateGOOD_A(int rows, int columns)
{
    bool[,] GoodA1 = GenerateA(rows, columns);
    bool[,] GoodA2 = new bool[rows, columns];
    int RANK = 0;

    if (rows >= columns)
    {
        while (RANK < columns)
        {
            GoodA1 = GenerateA(GoodA1.GetLength(0), GoodA1.GetLength(1));
            Array.Copy(GoodA1, GoodA2, rows * columns);
            RANK = Rank.Of(GoodA1);
        }
    }
    else
    {
        while (RANK < rows)
        {
            GoodA1 = GenerateA(GoodA1.GetLength(0), GoodA1.GetLength(1));
            Array.Copy(GoodA1, GoodA2, rows * columns);
            RANK = Rank.Of(GoodA1);
        }
    }
```

```
    return GoodA2;
}
private static bool[,] GenerateGOOD_M(int rows, int columns)
{
    bool[,] GoodM1 = GenerateM(rows, columns);
    bool[,] GoodM2 = new bool[rows, columns];
    int RANK = 0;
    if (rows >= columns)
    {
        while (RANK < columns)
        {
            GoodM1 = GenerateA(GoodM1.GetLength(0), GoodM1.GetLength(1));
            Array.Copy(GoodM1, GoodM2, rows * columns);
            RANK = Rank.Of(GoodM1);
        }
    }
    else
    {
        while (RANK < rows)
        {
            GoodM1 = GenerateA(GoodM1.GetLength(0), GoodM1.GetLength(1));
            Array.Copy(GoodM1, GoodM2, rows * columns);
            RANK = Rank.Of(GoodM1);
        }
    }
    return GoodM2;
}
public static bool[,] Good_RandomBool(int raws, int columns)
{
    Random r = new Random();
    bool[,] res = new bool[raws, columns];
    bool[,] res2 = new bool[raws, columns];

    Array.Copy(res, res2, raws * columns);

    int RANK = 0;
    if (raws >= columns)
    {
        while (RANK < columns)
        {
            for (int i = 0; i < raws; i++)
                for (int j = 0; j < columns; j++)
                {
                    if (r.Next(0, 2) == 1)
                    { res[i, j] = true; }
                    else
                    { res[i, j] = false; }
                }
            RANK = Rank.Of(res);
        }
    }
    else
    {
        while (RANK < raws)
        {
            for (int i = 0; i < raws; i++)
                for (int j = 0; j < columns; j++)
                {
                    if (r.Next(0, 2) == 1)
                    { res[i, j] = true; }
                    else
                    { res[i, j] = false; }
```

```
                    }
                RANK = Rank.Of(res);
            }
        }
        return res;
    }



    private static bool[,] ConvertMatrix(bool[][] matrix)
    {
        bool[,] res = new bool[matrix.Length, matrix[0].Length];

        for (int i = 0; i < matrix.Length; i++)
        {
            for (int j = 0; j < matrix[0].Length; j++)
            {

                res[i, j] = matrix[i][j];
            }
        }
        return res;
    }


    private static void MatrixFromFile(bool[,] matrix, string directory)
    {

        string[] mass = File.ReadAllLines(directory);
        List<string[]> list = new List<string[]>();

        char[] a = new char[] { ' ' };
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            list.Add(mass[i].Split(a));
        }
        string[][] o = list.ToArray();

        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            for (int j = 0; j < matrix.GetLength(1); j++)
            {
                matrix[i, j] = false;
                if (o[i][j] == "1")
                {
                    matrix[i, j] = true;
                }

            }
        }
    }
  }
}

//Файл Print.cs
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Checking
```

```
{
    class Print
    {
        public static void Bits(BitArray f)
        {
            int count = 0;
            for (int i = 0; i < f.Length; i++)
            {
                if (f[i] == true)
                { Console.Write("1"); }
                else
                { Console.Write("0"); }
                count++;

                if (count == 8)
                {
                    count = 0;
                    Console.Write(" ");
                }
            }
            Console.WriteLine();
        }
        public static void Matrix(int[,] res)
        {

            for (int i = 0; i < res.GetLength(0); i++)
            {
                for (int j = 0; j < res.GetLength(1); j++)
                {
                    Console.Write(res[i, j]);
                }
                Console.WriteLine();
            }
            Console.WriteLine();
        }
        public static void Matrix(byte[,] res)
        {

            for (int i = 0; i < res.GetLength(0); i++)
            {
                for (int j = 0; j < res.GetLength(1); j++)
                {
                    Console.Write("{0:x}" + " ", res[i, j]);
                }
                Console.WriteLine();
            }
            Console.WriteLine();
        }
        public static void Matrix(bool[,] res)
        {
            for (int i = 0; i < res.GetLength(0); i++)
            {
                for (int j = 0; j < res.GetLength(1); j++)
                {
                    if (res[i, j] == true)
                    { Console.Write(1); }
                    else
                    { Console.Write(0); }
                }
                Console.WriteLine();
            }
            Console.WriteLine();
```

```
        }
        public static void Array(byte[] array)
        {
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write("{0:x} ", array[i]);
            }
            Console.WriteLine();
        }
        public static void Array(bool[] array)
        {
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i])
                {
                    Console.Write(1);
                }
                else
                {
                    Console.Write(0);
                }
            }
            Console.WriteLine();
        }

        public static void Array(int[] array)
        {
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write(array[i] + " ");
            }
            Console.WriteLine();
        }
    }
}

//Файл Program.cs
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

namespace Checking
{
    static class Program
    {
        //считывание матрицы из текстового файла
        static void MatrixFromFile(bool[,] matrix, string directory)
        {

            string[] mass = File.ReadAllLines(directory);
            List<string[]> list = new List<string[]>();

            char[] a = new char[] { ' ' };
            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                list.Add(mass[i].Split(a));
            }
```

```
            string[][] o = list.ToArray();

            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(1); j++)
                {
                    matrix[i, j] = false;
                    if (o[i][j] == "1")
                    {
                        matrix[i, j] = true;
                    }

                }
            }
        }

        //запись матрицы в текстовый файл
        static void MatrixIntoFile(bool[,] matrix, string directory)
        {
            StreamWriter sw = new StreamWriter(directory);
            for (int i = 0; i < matrix.GetLength(0); i++)
            {
                for (int j = 0; j < matrix.GetLength(1); j++)
                {
                    if (matrix[i, j])
                    {
                        sw.Write(1 + " ");
                    }
                    else
                    {
                        sw.Write(0 + " ");
                    }
                }
                sw.Write(sw.NewLine);
            }
            sw.Close();
        }

        //вспомогательные функции для преобразования типов матриц
        static bool[,] MatrixToEvaluate(List<bool[,]> MA_List)
        {
            bool[,] MA_Array = new bool[MA_List[0].GetLength(0) * MA_List.Count,
MA_List[0].GetLength(1)];

            for (int k = 0; k < MA_List.Count; k++)
            {
                for (int i = 0; i < MA_List[0].GetLength(0); i++)
                {
                    for (int j = 0; j < MA_List[0].GetLength(1); j++)
                    {
                        MA_Array[i + MA_List[0].GetLength(0) * k, j] = MA_List[k][i, j];
                    }
                }
            }
            return MA_Array;
        }
        static int[,] BoolMatrixToInt(bool[,] A)
        {
            int[,] result = new int[A.GetLength(0), A.GetLength(1)];
            for (int i = 0; i < A.GetLength(0); i++)
            {
                for (int j = 0; j < A.GetLength(1); j++)
```

```
            {
                if (A[i, j] == true)
                {
                    result[i, j] = 1;
                }
                else
                {
                    result[i, j] = 0;
                }
            }
        }
        return result;
    }
    static bool[,] IntMatrixToBool(int[,] A)
    {
        bool[,] result = new bool[A.GetLength(0), A.GetLength(1)];
        for (int i = 0; i < A.GetLength(0); i++)
        {
            for (int j = 0; j < A.GetLength(1); j++)
            {
                if (A[i, j] == 1)
                {
                    result[i, j] = true;
                }
                else
                {
                    result[i, j] = false;
                }
            }
        }
        return result;
    }


    // умножение матриц
    static bool[,] Multiply_matrix(bool[,] matrix1, bool[,] matrix2)
    {
        if (matrix1.GetLength(1) != matrix2.GetLength(0))
            throw new ArgumentException("Invalid arrays length");
        int commonLength = matrix1.GetLength(1);

        bool[,] res = new bool[matrix1.GetLength(0), matrix2.GetLength(1)];

        for (int i = 0; i < res.GetLength(0); i++)
        {
            for (int j = 0; j < res.GetLength(1); j++)
            {
                bool nextVal = false;
                for (int k = 0; k < commonLength; k++)
                {
                    nextVal ^= matrix1[i, k] & matrix2[k, j];
                }
                res[i, j] = nextVal;
            }
        }
        return res;
    }
    static bool[,] Multiply_matrix_ext(this bool[,] matrix1, bool[,] matrix2)
    {
        if (matrix1.GetLength(1) != matrix2.GetLength(0))
            throw new ArgumentException("Invalid arrays length");
        int commonLength = matrix1.GetLength(1);
```

```
    bool[,] res = new bool[matrix1.GetLength(0), matrix2.GetLength(1)];

    for (int i = 0; i < res.GetLength(0); i++)
    {
        for (int j = 0; j < res.GetLength(1); j++)
        {
            bool nextVal = false;
            for (int k = 0; k < commonLength; k++)
            {
                nextVal ^= matrix1[i, k] & matrix2[k, j];
            }
            res[i, j] = nextVal;
        }
    }
    return res;
}

//возведение матрицы в степень
static bool[,] Matrix_degree(bool[,] matrix, int degree)
{
    if (degree == 0)
    { return new bool[,] { { false, false }, { false, false } }; }

    else
    {
        bool[,] tmp = matrix;
        for (int i = 1; i < degree; i++)
        {
            tmp = tmp.Multiply_matrix_ext(matrix);
        }
        return tmp;
    }
}

//генерирование случайной "хорошей" матрицы (полного ранга)
static bool[,] Good_RandomBool(int raws, int columns)
{
    Random r = new Random();
    bool[,] res = new bool[raws, columns];
    bool[,] res2 = new bool[raws, columns];
    Array.Copy(res, res2, raws * columns);
    int RANK = 0;
    if (raws >= columns)
    {
        while (RANK < columns)
        {
            for (int i = 0; i < raws; i++)
                for (int j = 0; j < columns; j++)
                {
                    if (r.Next(0, 2) == 1)
                    { res[i, j] = true; }
                    else
                    { res[i, j] = false; }
                }
            RANK = Rank.Of(res);
        }
    }
    else
    {
        while (RANK < raws)
        {
```

```
                for (int i = 0; i < raws; i++)
                    for (int j = 0; j < columns; j++)
                    {
                        if (r.Next(0, 2) == 1)
                        { res[i, j] = true; }
                        else
                        { res[i, j] = false; }

                    }
                RANK = Rank.Of(res);
            }
        }
        return res;

    }

    //генерирование случайной матрицы
    static bool[,] RandomBool(int raws, int columns)
    {
        Random r = new Random();
        bool[,] res = new bool[raws, columns];

        for (int i = 0; i < raws; i++)
            for (int j = 0; j < columns; j++)
            {

                if (r.Next(0, 2) == 1)
                { res[i, j] = true; }
                else
                { res[i, j] = false; }

            }

        return res;

    }


    //вспомогательные функции преобразования строки в столбец и наоборот, преобразования
типов и выборки "существенных" переменных
    static bool[,] Convertion(bool[] ArrayToConvert)
    {
        bool[,] result = new bool[ArrayToConvert.Length, 1];
        for (int i = 0; i < ArrayToConvert.Length; i++)
        {
            result[i, 0] = ArrayToConvert[i];
        }
        return result;
    }
    static bool[] Convertion(bool[,] ArrayToConvert)
    {
        bool[] result = new bool[ArrayToConvert.GetLength(0)];

        for (int i = 0; i < ArrayToConvert.Length; i++)
        {
            result[i] = ArrayToConvert[i, 0];
        }
        return result;
    }
    static int[] BoolToIntArray(bool[] ArrayToConvert)
    {
```

```
        int[] result = new int[ArrayToConvert.GetLength(0)];

        for (int i = 0; i < ArrayToConvert.Length; i++)
        {
            if (ArrayToConvert[i])
            {
                result[i] = 1;
            }
            else
            {
                result[i] = 0;
            }
        }
        return result;
    }
    static bool[] IntToBoolArray(int[] ArrayToConvert)
    {
        bool[] result = new bool[ArrayToConvert.GetLength(0)];

        for (int i = 0; i < ArrayToConvert.Length; i++)
        {
            if (ArrayToConvert[i] == 1)
            {
                result[i] = true;
            }
            else
            {
                result[i] = false;
            }
        }
        return result;
    }
    static bool[,] BitsToBool(BitArray bits)
    {
        bool[,] result = new bool[bits.Length, 1];

        for (int i = 0; i < bits.Count; i++)
        {
            result[i, 0] = bits.Get(i);
        }
        return result;
    }
    static BitArray choose_n(BitArray x, int[] F)
    {
        int n = F.Length;
        BitArray res = new BitArray(n);
        for (int i = 0; i < n; i++)
        {
            res.Set(i, x[F[i]]);
        }
        return res;
    }
    static bool[,] ConvertMatrix(bool[][] matrix)
    {
        bool[,] res = new bool[matrix.Length, matrix[0].Length];

        for (int i = 0; i < matrix.Length; i++)
        {
            for (int j = 0; j < matrix[0].Length; j++)
            {

                res[i, j] = matrix[i][j];
```

```
            }
        }
        return res;
    }

    //сравнение набора байтов
    static bool BitArrayEquality(BitArray A, BitArray B)
    {
        if (A.Length != B.Length)
        {
            return false;
        }
        int count = 0;
        for (int i = 0; i < A.Length; i++)
        {
            if (A.Get(i) != B.Get(i))
            {
                count++;
            }
        }

        if (count > 0)
        {
            return false;
        }
        else
        {
            return true;
        }

    }

    //генерирование матриц P и L
    static bool[,] GenerateP(int columns, string variables)
    {
        string f = variables;
        char[] param = new char[] { ',' };
        string[] splitted = f.Split(param);
        int[] FI = new int[columns];

        for (int i = 0; i < FI.Length; i++)
        {
            for (int j = 0; j < splitted.Length - 1; j++)
            {
                FI[i] = 0;
                FI[int.Parse(splitted[j])] = 1;
            }
        }
        bool[] l = new bool[FI.Length];
        bool[][] result = new bool[splitted.Length - 1][];

        for (int i = 0; i < FI.Length; i++)
        {
            if (FI[i] != 0)
            {
                l[i] = true;
            }
            else
            {
                l[i] = false;
            }
        }
```

```
            for (int k = 0; k < splitted.Length - 1; k++)
            {
                result[k] = new bool[columns];
                for (int i = 0; i < columns; i++)
                {
                    if (l[i] == true)
                    {
                        for (int g = 0; g < columns; g++)
                        {
                            result[k][g] = false;
                            result[k][i] = true;
                        }
                        l[i] = false;
                        break;
                    }
                }
            }
            return ConvertMatrix(result);
        }
        static bool[,] GenerateL(string polinom)
        {
            string f = polinom;
            char[] param = new char[] { ',' };
            string[] splitted = f.Split(param);
            int LFSR_lenght = int.Parse(splitted.Last());
            bool[][] result = new bool[LFSR_lenght - 1][];
            for (int i = 0; i < LFSR_lenght - 1; i++)
            {
                result[i] = new bool[LFSR_lenght];
                for (int j = 0; j < LFSR_lenght; j++)
                {
                    result[i][j] = false;
                    result[i][i + 1] = true;
                }
            }
            bool[] LastRow = new bool[int.Parse(splitted[splitted.Length - 1])];
            for (int i = 0; i < LastRow.Length; i++)
            {
                for (int j = 0; j < splitted.Length - 1; j++)
                {
                    LastRow[i] = false;
                    LastRow[int.Parse(splitted[j])] = true;
                }
            }
            List<bool[]> F = new List<bool[]>();
            for (int i = 0; i < LFSR_lenght - 1; i++)
            {
                F.Add(result[i]);
            }
            F.Add(LastRow);
            result = F.ToArray();
            return ConvertMatrix(result);
        }
        // реализация генератора гаммы (вычисление состояний регистра в определенных тактах
и генерирование гаммы)
        static BitArray GenerateNecessaryState(BitArray registr, int statenumber, int[]
feed)
        {
            BitArray tmp = new BitArray(registr);

            for (int i = 0; i < statenumber; i++)
            {
```

```
                tmp = CircleShiftLeft(tmp, feed);
            }
            return tmp;
        }
        static BitArray CircleShiftLeft(BitArray registr, int[] feed)
        {
            BitArray tmp = new BitArray(registr);
            BitArray result = new BitArray(tmp.Length);
            for (int i = 0; i < tmp.Length - 1; i++)
            {
                result[i] = tmp[i + 1];
            }
            result[tmp.Length - 1] = CountC(tmp, feed);
            return result;
        }
        static bool CountC(BitArray register, int[] feed)
        {
            BitArray feedback = new BitArray(feed.Length);
            for (int i = 0; i < feedback.Length; i++)
            {
                feedback[i] = register[feed[i]];
            }
            bool c = feedback[0];
            for (int i = 1; i < feedback.Length; i++)
            {
                c ^= feedback[i];
            }
            return c;
        }
        static BitArray Generate_gamma(int bit_lenght, BitArray state, bool[,] M, int[] F,
int[] feed, double p)
        {
            BitArray result = new BitArray(bit_lenght);
            BitArray tmp = new BitArray(state);
            BitArray Mx = new BitArray(Convertion(Multiply_matrix(M,
BitsToBool(choose_n(tmp, F)))));
            for (int j = 0; j < bit_lenght; j++)
            {
                result.Set(j, FI_two_variables_with_Random(Mx, p));
                tmp = GenerateNecessaryState(tmp, 1, feed);
                Mx = new BitArray(Convertion(Multiply_matrix(M, BitsToBool(choose_n(tmp,
F)))));
            }
            return result;
        }
        static BitArray Generate_gamma_clear(int bit_lenght, BitArray state, bool[,] M,
int[] F, int[] feed)
        {
            BitArray result = new BitArray(bit_lenght);
            BitArray tmp = new BitArray(state);
            BitArray Mx = new BitArray(Convertion(Multiply_matrix(M,
BitsToBool(choose_n(tmp, F)))));
            for (int j = 0; j < bit_lenght; j++)
            {
                result.Set(j, FI_two_variables(Mx));
                tmp = GenerateNecessaryState(tmp, 1, feed);
                Mx = new BitArray(Convertion(Multiply_matrix(M, BitsToBool(choose_n(tmp,
F)))));
            }

            return result;
        }
```

```
        //реализация исходной функции усложнения и приближающей функции, находящейся на
допустимом расстоянии
        static bool FI_two_variables(BitArray vectorX)
        {
            // функция приближения x0_&_x1
            return (vectorX.Get(0) & vectorX.Get(1));
        }
        static bool FI_two_variables_with_Random(BitArray vectorX, double p)
        {
            // функция приближения x0_&_x1
            bool random = RandomValue(p);
            return (vectorX.Get(0) & vectorX.Get(1)) ^ random;
        }
        static bool RandomValue(double p)
        {
            Thread.Sleep(10);
            Random r = new Random();
            int psi = r.Next(int.MaxValue);
            double x = (int.MaxValue * p) - 1;
            if (psi <= x)
            {
                return true;
            }
            else
            {
                return false;
            }
        }
        //построение таблицы векторов в лексикографическом порядке
        static BitArray[] VectorTable(int s)
        {
            int rows = (int)Math.Pow((double)2, (double)s);
            bool[] tmp = new bool[s];
            BitArray[] result = new BitArray[rows];
            string x = string.Empty;
            char[] characters = new char[0];
            bool[] vector = new bool[0];
            for (int i = 0; i < rows; i++)
            {
                x = Convert.ToString(i, 2);
                x = x.PadLeft(s, '0');
                characters = x.ToCharArray();
                vector = new bool[characters.Length];
                for (int j = 0; j < characters.Length; j++)
                {
                    if (characters[j] == '1')
                    {
                        vector[j] = true;
                    }
                    else
                    {
                        vector[j] = false;
                    }
                }
                result[i] = new BitArray(vector);
                //result[i].CopyTo(tmp, 0);
                //Array.Reverse(tmp);
                //result[i] = new BitArray(tmp);
            }
            return result;
        }
```

```
//вспомогательные функции для очистки, записи\считывания данных из файлов
static void intoFile(BitArray bits, string DIR)
{
    StreamWriter sw = File.AppendText(DIR);
    for (int i = 0; i < bits.Length; i++)
    {
        if (bits.Get(i) == true)
        {
            sw.Write("1");
        }
        else
        {
            sw.Write("0");
        }
    }
    sw.WriteLine();
    sw.Close();
}
static void Clear(string DIR)
{
    StreamWriter sw = new StreamWriter(DIR);
    sw.Write(string.Empty);
    sw.Close();
}
static BitArray[] fromFile(string DIR)
{
    string[] rows = File.ReadAllLines(DIR);

    bool[] result = new bool[rows[0].Length];

    BitArray[] bits = new BitArray[rows.Length];

    for (int j = 0; j < rows.Length; j++)
    {
        for (int i = 0; i < rows[0].Length; i++)
        {
            if (rows[j][i] == '0')
            {
                result[i] = false;
            }
            else
            {
                result[i] = true;
            }
        }
        bits[j] = new BitArray(result);
    }
    return bits;
}

//измерение времени выполнения участков кода
static void Time()
{
    Console.Write(DateTime.Now.Hour + ":");
    Console.Write(DateTime.Now.Minute + ":");
    Console.WriteLine(DateTime.Now.Second);
}

//глобальные межклассовые переменные, участвующие в вычислениях
public static List<bool[,]> MAi;
public static List<bool[,]> MBi;
```

```
        struct TESTING
        {
            public static int countTRUE = 0;
            public static int countFALSE = 0;
        }

        //демонстрация атаки
        static void Test_s_2(double epsilon, double p)
        {
            Clear(Directory.GetCurrentDirectory() + "/material/gammaF(x).txt");
            Clear(Directory.GetCurrentDirectory() + "/material/gammaG(x).txt");
            Clear(Directory.GetCurrentDirectory() + "/material/iv.txt");
            bool[,] A = new bool[128, 128];
            MatrixFromFile(A, Directory.GetCurrentDirectory() +
"/Example_1/A_matrix_128x128.txt");
            bool[,] B = new bool[128, 128];
            MatrixFromFile(B, Directory.GetCurrentDirectory() +
"/Example_1/B_matrix_128x128.txt");
            bool[,] M = new bool[2, 64];
            MatrixFromFile(M, Directory.GetCurrentDirectory() +
"/Example_1/M_matrix_2x64.txt");
            string variables =
"0,1,2,4,5,6,7,11,13,15,18,19,20,22,24,27,29,30,33,38,39,40,41,42,48,49,50,54,59,60,61,62,63
,64,69,72,75,77,80,81,83,84,85,88,90,91,92,93,96,99,100,101,102,103,104,105,109,112,118,119,
121,122,124,126,128";
            string polinom = "0,36,38,45,57,95,128";
            bool[,] P = GenerateP(128, variables);
            bool[,] L = GenerateL(polinom);
            bool[,] key = new bool[128, 1];
            MatrixFromFile(key, Directory.GetCurrentDirectory() + "/Example_1/key.txt");
            bool[,] iv = new bool[128, 1];
            MatrixFromFile(iv, Directory.GetCurrentDirectory() + "/Example_1/iv.txt");
            intoFile(new BitArray(Convertion(iv)), Directory.GetCurrentDirectory() +
"/material/iv.txt");
            int[] F = new int[] { 0, 1, 2, 4, 5, 6, 7, 11, 13, 15, 18, 19, 20, 22, 24, 27,
29, 30, 33, 38, 39, 40, 41, 42, 48, 49, 50, 54, 59, 60, 61, 62, 63, 64, 69, 72, 75, 77, 80,
81, 83, 84, 85, 88, 90, 91, 92, 93, 96, 99, 100, 101, 102, 103, 104, 105, 109, 112, 118,
119, 121, 122, 124, 126 };
            int[] feed = new int[] { 0, 36, 38, 45, 57, 95 }
            MAi = new List<bool[,]>();
            MBi = new List<bool[,]>();
            BitArray state = new BitArray(128);
            BitArray Ak = new BitArray(128);
            BitArray Bc = new BitArray(128);

            int N = 128;
            int l0 = 128;
            int l1 = 128;
            int n = 64;
            int s = 2;
            int spow2 = (int)Math.Pow(2, (double)s);
            double delta = 0.01
            Console.WriteLine("Исходные параметры:");
            Console.WriteLine("Длина регистра - N = {0}", N);
            Console.WriteLine("Длина ключа - l0 = {0}", l0);
            Console.WriteLine("Длина вектора инициализации - l1 = {0}", l1);
            Console.WriteLine("Количество переменных функции усложнения - n = {0}", n);

            Console.WriteLine("Функция приближения - конъюнкция");

            Console.WriteLine("Количество переменных приближающей функции - s = {0}", s);
```

```
            Console.WriteLine("Вероятность ошибки алгоритма восстановления ключа - delta =
{0}", delta);
            Console.WriteLine("Степень близости приближающей функции - epsilon = {0}",
epsilon);
            Console.WriteLine("Вероятность совпадения результатов исходной функции и
приближающей - p = {0} (p >= 1/2*(1 + epsilon)", 1 - p);
            Time();
            Console.WriteLine("Этап №1 - Формирование множества I....");
            int[] I = Stage_1_Build_I.I(N, l0, l1, n, s, polinom, variables);
            Time();
            int r = (int)Math.Floor(8 * Math.Pow(epsilon, -2) * (Math.Log((Math.Pow(2,
(double)s) * I.Length * Math.Pow(delta, -1))))) + 1;
            Console.WriteLine(r);
            r = 172;
            Console.WriteLine("Число перезапусков генератора r = {0} (объем выборки
материала)", r);
            Console.WriteLine("Генерирование гаммы с заданными исходными параметрами...");
            BitArray Fx = new BitArray(1);
            BitArray Gx = new BitArray(1);

            BitArray LAk = new BitArray(1);
            BitArray LBc = new BitArray(1);

            for (int j = 0; j < r; j++)
            {
                Ak = new BitArray(Convertion(Multiply_matrix(A, key)));
                Bc = new BitArray(Convertion(Multiply_matrix(B, iv)));
                state = Ak.Xor(Bc);
                Fx = Generate_gamma(I.Last() + 1, state, M, F, feed, p);
                iv = RandomBool(128, 1);
                Gx = Generate_gamma_clear(I.Last() + 1, state, M, F, feed);
                intoFile(Fx, Directory.GetCurrentDirectory() + "/material/gammaF(x).txt");
                intoFile(Gx, Directory.GetCurrentDirectory() + "/material/gammaG(x).txt");
                intoFile(new BitArray(Convertion(iv)), Directory.GetCurrentDirectory() +
"/material/iv.txt");
            }
            BitArray[] ivs = fromFile(Directory.GetCurrentDirectory() + "/material/iv.txt");
            BitArray[][] MBiCj = new BitArray[I.Length][];
            if (I[0] == 0)
            {
                MBiCj[0] = new BitArray[r];
                for (int j = 0; j < r; j++)
                {
                    MBiCj[0][j] = new BitArray(Convertion(Multiply_matrix(MBi[0],
BitsToBool(ivs[j]))));
                }
                ParallelLoopResult loopResult = Parallel.For(
                     1,
                     I.Length,
                     (i, loopState) =>
                     {

                         MBiCj[i] = new BitArray[r];
                         for (int j = 0; j < r; j++)
                         {
                             MBiCj[i][j] = new BitArray(Convertion(Multiply_matrix(MBi[i],
BitsToBool(ivs[j]))));
                         }
                     }
                 );
            }
            else
```

```
            {
                ParallelLoopResult loopResult = Parallel.For(
                     0,
                     I.Length,
                     (i, loopState) =>
                     {
                         MBiCj[i] = new BitArray[r];
                         for (int j = 0; j < r; j++)
                         {
                             MBiCj[i][j] = MBiCj[i][j] = new
BitArray(Convertion(Multiply_matrix(MBi[i], BitsToBool(ivs[j]))));
                         }
                     }
                 );
            }
            Time();
            BitArray[] y = VectorTable(2);
            BitArray[] gamma = fromFile(Directory.GetCurrentDirectory() +
"/material/gammaF(x).txt");
            Console.WriteLine("Этап №2 - Работа метода максимума правдоподобия");
            int[][] counts = new int[I.Length][];
            for (int i = 0; i < counts.Length; i++)
            {
                counts[i] = new int[spow2];
            }
            BitArray tmp = new BitArray(s);
            for (int i = 0; i < I.Length; i++)
            {
                for (int k = 0; k < spow2; k++)
                {
                    for (int j = 0; j < r; j++)
                    {
                        tmp = new BitArray(y[k]);
                        if
(!FI_two_variables(tmp.Xor(MBiCj[i][j])).Equals(gamma[j].Get(I[i])))
                        {
                            counts[i][k]++;
                        }
                    }
                }
            }
            List<bool[,]> vector = new List<bool[,]>();
            bool[,] vector_part = new bool[2, 1];
            int min = 0;
            for (int i = 0; i < I.Length; i++)
            {
                min = counts[i].Min();
                for (int j = 0; j < spow2; j++)
                {
                    if (counts[i][j] == min)
                    {
                        vector_part = BitsToBool(y[j]);
                        vector.Add(vector_part);
                    }
                }
            }
            int[,] matrix = BoolMatrixToInt(MatrixToEvaluate(MAi));
            bool[] full_vector = Convertion(MatrixToEvaluate(vector));

            if (BitArrayEquality(new BitArray(full_vector), new
BitArray(Convertion(Multiply_matrix(MatrixToEvaluate(MAi), key)))))
            {
```

```
                TESTING.countTRUE++;
            }
            else
            {
                TESTING.countFALSE++;
            }
            Time();
        }
        static void Main(string[] args)
        {
            Test_s_2(0.3, 0.35);
            Console.ReadLine();
        }
    }
}

//Файл Rank.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Checking
{
    class Rank
    {
        // поиск булевого ранга матрицы
        public static int Of(bool[,] inp_Matr)
        {
            bool[,] buffer = new bool[inp_Matr.GetLength(0), inp_Matr.GetLength(1)];
            Array.Copy(inp_Matr, buffer, inp_Matr.GetLength(0) * inp_Matr.GetLength(1));
            int count = 0;
            if (buffer.GetLength(0) > 1 && buffer.GetLength(1) > 1)
            {
                ToOneBool(0, 0, buffer, false);
            }
            else
            {
                int end = buffer.GetLength(0) < buffer.GetLength(1) ? buffer.GetLength(0) :
buffer.GetLength(1);
                if (buffer.GetLength(0) < buffer.GetLength(1))
                {
                    for (int i = 0; i < buffer.GetLength(1); i++)
                    {
                        if (buffer[0, i] != false) { count++; break; }
                    }
                    return count;
                }
                else
                {
                    for (int i = 0; i < buffer.GetLength(0); i++)
                    {
                        if (buffer[i, 0] != false) { count++; break; }
                    }
                    return count;
                }
            }
            for (int i = 0; i < buffer.GetLength(0); i++)
            {
                for (int j = 0; j < buffer.GetLength(1); j++)
                {
                    if (buffer[i, j] != false) { count++; break; }
```

```
                }

            }
            return count;
        }
        internal static void ToOneBool(int row, int tree, bool[,] arr, bool flag)
        {
            int j = tree;
            /*ищем не нулевой елемент в первом столбце, если этот елемент является
             первым, переходим к шагу 1.1, в противном случае меняем строку с не
             нулевым елементом местами с первой строкой-> к шагу 1.1, если елемент
             не найден, отбрасываем первый столбец*/
            for (int i = row; i < arr.GetLength(0); i++)//1 часть
            {
                if (arr[i, j] != false && flag == false)
                {
                    if (i == row) break;
                    for (int w = tree; w < arr.GetLength(1); w++)
                    {
                        swap(row, w, i, w, arr);
                    }
                    flag = true;
                    break;
                }
                else if (flag == false && tree < arr.GetLength(1) - 1 && i ==
arr.GetLength(0) - 1)
                {
                    ToOneBool(row, tree + 1, arr, false);
                }
            }
            /*если 'элемент проверяемого столбца ненулевой - складываем по модулю 2 первую
строку и строку с ненулевым элементом */
            for (int i = row + 1; i < arr.GetLength(0); i++)
            {//2 часть
                if (arr[i, tree] == true)
                {
                    for (int tr = tree; tr < arr.GetLength(1); tr++)
                    {
                        arr[i, tr] = arr[i, tr] ^ arr[row, tr];
                    }
                }
            }
            /*Отбрасываем первый столбец и первую строку(следовательно вторая строка
становится первой по счету и второй столбец тоже) -> к шагу 1*/
            if (row < arr.GetLength(0) - 1 && tree < arr.GetLength(1) - 1)
            {
                ToOneBool(row + 1, tree + 1, arr, false);
            }
            /*else//если дошли до конца - ничего не меняем
            {
                arr[row, tree] = Math.Abs(arr[row, tree]);
            }*/
        }
        private static void swap(int x1, int y1, int x2, int y2, bool[,] arr)
        {
            arr[x1, y1] ^= arr[x2, y2];
            arr[x2, y2] = arr[x1, y1] ^ arr[x2, y2];
            arr[x1, y1] ^= arr[x2, y2];
        }
    }
}
```

```
//Файл Stage_1_Build_I.cs
using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;

namespace Checking
{
    static class Stage_1_Build_I
    {
        public static List<bool[,]> MAList;
        public static List<bool[,]> MBList;

        private static List<bool[,]> PLBList;
        private static int rank;


        // главный метод построения множества I
        public static int[] I(int N, int l0, int l1, int n, int s, string StreamKey, string
variables)
        {
            Matrix.Set(N,l0,l1,s,n,StreamKey,variables);

            bool[,] P = Matrix.P;
            bool[,] L = Matrix.L;
            bool[,] B = Matrix.B;
            bool[,] A = Matrix.A;
            bool[,] M = Matrix.M;

            //Console.WriteLine(P.GetLength(0));
            //Print.Matrix(P);
            //Console.ReadLine();

            //Console.WriteLine("Новые матрицы сгенерированы");

            int[] I = new int[200];
            bool[,] PLB;

            MAList = new List<bool[,]>();
            MBList = new List<bool[,]>();
            PLBList = new List<bool[,]>();

            int i = 0;
            int count = 0;

            do
            {
                //Console.WriteLine(i);
                if (Matrix_degree(L, i).GetLength(0) == 2)
                {
                    PLB = Multiply_matrix(P, B);
                }
                else
                {
                    PLB = GeneratePLB(P, Matrix_degree(L, i), B);
                }

                //Console.ReadLine();
                //Console.WriteLine(PLB.GetLength(0));
                //Console.WriteLine(PLB.GetLength(1));
```

```
            //Console.WriteLine(Rank.Of(PLB));

            if (Rank.Of(PLB) == n)
            {
                if (Matrix_degree(L, i).GetLength(0) == 2)
                {
                    MAList.Add(GenerateMA(M, Multiply_matrix(P, A)));
                }
                else
                {
                    MAList.Add(GenerateMA(M, GeneratePLA(P, Matrix_degree(L, i), A)));
                }

                MBList.Add(GenerateMB(M, PLB));

                rank = Rank.Of(MatrixToEvaluate(MAList));
                //Console.WriteLine(rank);
                I[count] = i;
                count++;
            }
            i++;
            if (rank == N)
            {
                break;
            }
        }
        while (rank <= N);
        Array.Resize(ref I, count);


        Program.MAi = MAList;
        Program.MBi = MBList;
        return I;

    }
    static bool[,] Convertion(bool[] ArrayToConvert)
    {
        bool[,] result = new bool[ArrayToConvert.Length, 1];
        for (int i = 0; i < ArrayToConvert.Length; i++)
        {
            result[i, 0] = ArrayToConvert[i];
        }
        return result;
    }
    static bool[] Convertion(bool[,] ArrayToConvert)
    {
        bool[] result = new bool[ArrayToConvert.GetLength(0)];

        for (int i = 0; i < ArrayToConvert.Length; i++)
        {
            result[i] = ArrayToConvert[i, 0];
        }
        return result;
    }
    static bool[,] BitsToBool(BitArray bits)
    {
        bool[,] result = new bool[bits.Length, 1];

        for (int i = 0; i < bits.Count; i++)
        {
            result[i, 0] = bits.Get(i);
        }
```

```
            return result;
        }
        // методы генерации вспомогательных матриц, использующихся в вычислениях
        private static bool[,] GeneratePLB(bool[,] P, bool[,] L1, bool[,] B)
        {
            bool[,] result = new bool[P.GetLength(0), B.GetLength(1)];
            result = Multiply_matrix(Multiply_matrix(P, L1), B);
            return result;

        }
        private static bool[,] GenerateMB(bool[,] M, bool[,] PLB)
        {
            bool[,] result = new bool[M.GetLength(0), PLB.GetLength(1)];
            result = Multiply_matrix(M, PLB);
            return result;
        }
        private static bool[,] GeneratePLA(bool[,] P, bool[,] L1, bool[,] A)
        {
            bool[,] result = new bool[P.GetLength(0), A.GetLength(1)];
            result = Multiply_matrix(Multiply_matrix(P, L1), A);
            return result;

        }

        private static bool[,] GenerateMA(bool[,] M, bool[,] PLA)
        {
            bool[,] result = new bool[M.GetLength(0), PLA.GetLength(1)];
            result = Multiply_matrix(M, PLA);
            return result;
        }
        // операции с матрицами, необходимые для вычисления множества I, удовлетворяющего
условию 2
        public static bool[,] Matrix_degree(bool[,] matrix, int degree)
        {
            if (degree == 0)
            { return new bool[,] { { false, false }, { false, false } }; }

            else
            {
                bool[,] tmp = matrix;
                for (int i = 1; i < degree; i++)
                {
                    tmp = tmp.Multiply_matrix(matrix);
                }
                return tmp;
            }
        }

        public static bool[,] MatrixToEvaluate(List<bool[,]> MA_List)
        {
            bool[,] MA_Array = new bool[MA_List[0].GetLength(0) * MA_List.Count,
MA_List[0].GetLength(1)];

            for (int k = 0; k < MA_List.Count; k++)
            {
                for (int i = 0; i < MA_List[0].GetLength(0); i++)
                {
                    for (int j = 0; j < MA_List[0].GetLength(1); j++)
                    {
                        MA_Array[i + MA_List[0].GetLength(0) * k, j] = MA_List[k][i, j];
                    }
                }
```

```
        }
        return MA_Array;
    }
    public static bool[,] Multiply_matrix(this bool[,] matrix1, bool[,] matrix2)
    {
        if (matrix1.GetLength(1) != matrix2.GetLength(0))
            throw new ArgumentException("Invalid arrays length");
        int commonLength = matrix1.GetLength(1);

        bool[,] res = new bool[matrix1.GetLength(0), matrix2.GetLength(1)];

        for (int i = 0; i < res.GetLength(0); i++)
        {
            for (int j = 0; j < res.GetLength(1); j++)
            {
                bool nextVal = false;
                for (int k = 0; k < commonLength; k++)
                {
                    nextVal ^= matrix1[i, k] & matrix2[k, j];
                }
                res[i, j] = nextVal;
            }
        }
        return res;
    }
    public static int[,] Multiply_matrix(this int[,] matrix1, int[,] matrix2)
    {

        if (matrix1.GetLength(1) != matrix2.GetLength(0))
            throw new ArgumentException("Invalid arrays length");
        int commonLength = matrix1.GetLength(1);

        int[,] res = new int[matrix1.GetLength(0), matrix2.GetLength(1)];

        for (int i = 0; i < res.GetLength(0); i++)
        {
            for (int j = 0; j < res.GetLength(1); j++)
            {
                int nextVal = 0;
                for (int k = 0; k < commonLength; k++)
                {
                    nextVal ^= matrix1[i, k] & matrix2[k, j];
                }
                res[i, j] = nextVal;
            }
        }
        return res;
    }
    static int[,] BoolMatrixToInt(bool[,] A)
    {
        int[,] result = new int[A.GetLength(0), A.GetLength(1)];
        for (int i = 0; i < A.GetLength(0); i++)
        {
            for (int j = 0; j < A.GetLength(1); j++)
            {
                if (A[i, j] == true)
                {
                    result[i, j] = 1;
                }
                else
                {
                    result[i, j] = 0;
```

```
                    }
                }
            }
            return result;
        }

    }
}

//Файл Stage_3_LinearSolve.cs
using System;
using System.Collections;
using System.Data;

namespace Checking
{
    public class GaussSolutionNotFound : Exception
    {
        public GaussSolutionNotFound(string msg)
            : base("Решение не может быть найдено: \r\n" + msg)
        {
        }
    }

    public class Stage_3
    {
        private int[,] initial_a_matrix;
        //private int[,] a_matrix;  // матрица A
        private int[,] a_matrix;  // матрица A
        private int[] x_vector;   // вектор неизвестных x
        private int[] initial_b_vector;
        private int[] b_vector;   // вектор b
        private int[] u_vector;   // вектор невязки U
        private int eps;          // порядок точности для сравнения вещественных чисел
        private int size;            // размерность задачи


        /// <summary>
        /// булева матрица и булев вектор
        /// </summary>
        /// <param name="a_matrix"></param>
        /// <param name="b_vector"></param>
        public Stage_3(int[,] a_matrix, int[] b_vector, int eps)
        {
            if (a_matrix == null || b_vector == null)
                throw new ArgumentNullException("Один из параметров равен null.");

            int b_length = b_vector.Length;
            int a_length = a_matrix.Length;
            if (a_length != b_length * b_length)
                throw new ArgumentException(@"Количество строк и столбцов в матрице A должно
совпадать с количеством элементров в векторе B.");

            this.initial_a_matrix = a_matrix;  // запоминаем исходную матрицу
            this.a_matrix = (int[,])a_matrix.Clone(); // с её копией будем производить
вычисления
            this.initial_b_vector = b_vector;  // запоминаем исходный вектор
            this.b_vector = (int[])b_vector.Clone();  // с его копией будем производить
вычисления
            this.x_vector = new int[b_length];
            this.u_vector = new int[b_length];
            this.size = b_length;
```

```
            this.eps = eps;

            GaussSolve();
        }

        public int[] XVector
        {
            get
            {
                return x_vector;
            }
        }

        public int[] UVector
        {
            get
            {
                return u_vector;
            }
        }

        // инициализация массива индексов столбцов
        private int[] InitIndex()
        {
            int[] index = new int[size];
            for (int i = 0; i < index.Length; ++i)
                index[i] = i;
            return index;
        }

        // поиск главного элемента в матрице
        private int FindR(int row, int[] index)
        {
            int max_index = row;
            int max = a_matrix[row, index[max_index]];
            int max_abs = Math.Abs(max);
            //if(row < size - 1)
            for (int cur_index = row + 1; cur_index < size; ++cur_index)
            {
                int cur = a_matrix[row, index[cur_index]];
                int cur_abs = Math.Abs(cur);
                if (cur_abs > max_abs)
                {
                    max_index = cur_index;
                    max = cur;
                    max_abs = cur_abs;
                }
            }

            if (max_abs < eps)
            {
                if (Math.Abs(b_vector[row]) > eps)
                    throw new GaussSolutionNotFound("Система уравнений несовместна.");
                else
                    throw new GaussSolutionNotFound("Система уравнений имеет множество
решений.");
            }

            // меняем местами индексы столбцов
            int temp = index[row];
            index[row] = index[max_index];
            index[max_index] = temp;
```

```
    return max;
}

// Нахождение решения СЛУ методом Гаусса
private void GaussSolve()
{
    int[] index = InitIndex();
    GaussForwardStroke(index);
    GaussBackwardStroke(index);
    GaussDiscrepancy();
}

// Прямой ход метода Гаусса
private void GaussForwardStroke(int[] index)
{
    // перемещаемся по каждой строке сверху вниз
    for (int i = 0; i < size; ++i)
    {
        // 1) выбор главного элемента
        int r = FindR(i, index);

        // 2) преобразование текущей строки матрицы A
        for (int j = 0; j < size; ++j)
            a_matrix[i, j] &= r;

        // 3) преобразование i-го элемента вектора b
        b_vector[i] &= r;

        // 4) Вычитание текущей строки из всех нижерасположенных строк
        for (int k = i + 1; k < size; ++k)
        {
            int p = a_matrix[k, index[i]];
            for (int j = i; j < size; ++j)
                a_matrix[k, index[j]] ^= a_matrix[i, index[j]] & p;
            b_vector[k] ^= b_vector[i] & p;
            a_matrix[k, index[i]] = 0;
        }
    }
}

// Обратный ход метода Гаусса
private void GaussBackwardStroke(int[] index)
{
    // перемещаемся по каждой строке снизу вверх
    for (int i = size - 1; i >= 0; --i)
    {
        // 1) задаётся начальное значение элемента x
        int x_i = b_vector[i];

        // 2) корректировка этого значения
        for (int j = i + 1; j < size; ++j)
            x_i ^= x_vector[index[j]] & a_matrix[i, index[j]];
        x_vector[index[i]] = x_i;
    }
}

// Вычисление невязки решения
// U = b - x * A
// x - решение уравнения, полученное методом Гаусса
private void GaussDiscrepancy()
{
```

```
                for (int i = 0; i < size; ++i)
                {
                    int actual_b_i = 0;   // результат перемножения i-строки
                    // исходной матрицы на вектор x
                    for (int j = 0; j < size; ++j)
                        actual_b_i ^= initial_a_matrix[i, j] & x_vector[j];
                    // i-й элемент вектора невязки
                    u_vector[i] = initial_b_vector[i] - actual_b_i;
                }
            }
        }
    }
```

Додаток Б

Программний код реалізації методу побудови списку високоймовірних наближень булевих функцій

Програмна реалізація методу побудови списку високоймовірних наближень булевих функцій виконана на ПК з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам’яті 4 Гб RAM на базі 32-розрядної ОС Windows 7 Service Pack 1. Мова програмування – C# .NET Framework 4.0. Середовище розробки – Microsoft Visual Studio 2010.

```
//файл G_Function.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
using Org.BouncyCastle.Math;

namespace Approximations
{
    public class G_Function
    {
        private int[,] A;

        private string Fi_vector;

        private string G_vector;

        private double distance;


        private int k;
        private int n;

        public G_Function(int n, int k, int[,] A, string Fi_vector)
        {
            this.n = n;
            this.k = k;
            this.A = A;
            this.Fi_vector = Fi_vector;
            G_vector = Build_G_Function(k, Operations.VectorTable(n, true), Fi_vector, A);
        }

        public double Evaluate_Distance(string Input_Function_Vector)
        {
            double result = (double)(new BigInteger(Input_Function_Vector, 2)).Xor(new
BigInteger(G_vector, 2)).ToString(2).Count(x => x == '1') / (double)G_vector.Length;
            this.distance = result;
            return result;
```

```
        }


        private static string Build_G_Function(int k, List<bool[,]> F_table, string
Fi_vector, int[,] A)
        {
            string G_function = "";

            BitArray EvaluateVector = new BitArray(k);

            for (int i = 0; i < F_table.Count; i++)
            {
                EvaluateVector =
Operations.BoolRowToBits(Operations.Multiply_matrix(F_table[i],
Operations.IntMatrixToBool(A)));
                if (Fi_vector[Operations.Int(EvaluateVector)] == '1')
                {
                    G_function = G_function + "1";
                }
                else
                {
                    G_function = G_function + "0";
                }
            }
            return G_function;
        }



        public double Count_Treshold_Distance(double epsilon)
        {
            return Math.Pow(2, -k) * (1 - epsilon);
        }

        public string Make_Polinom()
        {
            if (k > 10)
            {
                throw new ArgumentException("k must be less than 10. Or rewrite this
method");
            }

            if (Math.Log((double)Fi_vector.Length, 2) != k)
            {
                throw new ArgumentException("Check k!!!");
            }

            string polinom = "";
            string coefficients = Zhegalkin_Coefficients(Fi_vector);

            BitArray[] table = Operations.VectorTable(k);

            string[] x = new string[] { "1", "x1", "x2", "x3", "x4", "x5", "x6", "x7", "x8",
"x9" };

            string[] conjunctions = new string[(int)Math.Pow(2, k)];


            string one_conjunction = "";


            conjunctions[0] = "1";
```

```
        for (int i = 1; i < Math.Pow(2, k); i++)
        {
            for (int j = 0; j < k; j++)
            {
                if (table[i][j] == true)
                {
                    one_conjunction += "x" + (j + 1);
                }
            }
            conjunctions[i] = one_conjunction;
            one_conjunction = "";
        }

        for (int i = 0; i < Fi_vector.Length; i++)
        {
            if (coefficients[i] == '1')
            {
                polinom = polinom + conjunctions[i] + " + ";
            }
        }
        return polinom.Substring(0, polinom.Length - 2);
    }
    private static string Zhegalkin_Coefficients(string vector)
    {
        if (vector.Length % 2 != 0)
        {
            throw new ArgumentException("Incorrect vector!!! Chech length!!!");
        }

        StringBuilder result = new StringBuilder();
        List<string[]> x = new List<string[]>();

        x.Add(step(vector));

        for (int j = 0; j < (vector.Length - 2) / 2; j++)
        {
            for (int k = 0; k < 2; k++)
            {
                x.Add(step(x[j][k]));
            }
        }

        x = x.Where(q => string.Concat(q).Length == 2).ToList();

        foreach (string[] item in x)
        {
            result.Append(string.Concat(item));
        }

        return result.ToString();
    }
    private static string[] step(string vector)
    {
        if (vector.Length <= 1)
        {
            throw new ArgumentException("Check Vector!!!");
        }



        string one = vector.Substring(0, vector.Length / 2);
        string two = vector.Substring(vector.Length / 2, vector.Length / 2);
```

```
            string XOR = new BigInteger(one, 2).Xor(new BigInteger(two, 2)).ToString(2);

            if (XOR.Length != one.Length)
            {
                XOR = XOR.PadLeft(one.Length, '0');
            }
            return new string[] { one, XOR };

        }


        public void View_G_Function()
        {
            Console.WriteLine("A Matrix:");
            Print.Matrix(A);
            Console.WriteLine("Fi vector: {0}", Fi_vector);
            Console.WriteLine("Polinomial representation of Fi: {0}", Make_Polinom());
            Console.WriteLine("G vector : {0}", G_vector);

        }

    }
}


//файл Operations.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

namespace Approximations
{


    public static class Operations
    {
        //транспонирование матрицы n * k в k * n
        public static int[,] TransposeMatrix_N_K_to_K_N(int[,] matrix)
        {
            int[,] result = new int[matrix.GetLength(1), matrix.GetLength(0)];

            for (int i = 0; i < matrix.GetLength(1); i++)
            {
                for (int j = 0; j < matrix.GetLength(0); j++)
                {
                    result[i, j] = matrix[j, i];
                }
            }
            return result;
        }
        //логическое преобразование матрицы
        public static bool[,] IntMatrixToBool(int[,] A)
        {
            bool[,] result = new bool[A.GetLength(0), A.GetLength(1)];
            for (int i = 0; i < A.GetLength(0); i++)
            {
                for (int j = 0; j < A.GetLength(1); j++)
                {
                    if (A[i, j] == 1)
                    {
```

```
                    result[i, j] = true;
                }
                else
                {
                    result[i, j] = false;
                }
            }
        }
        return result;
    }

    //построение таблицы векторов в лексикографическом порядке
    public static BitArray[] VectorTable(int k)
    {
        int rows = (int)Math.Pow((double)2, (double)k);
        bool[] tmp = new bool[k];
        BitArray[] result = new BitArray[rows];
        string x = string.Empty;
        char[] characters = new char[0];
        bool[] vector = new bool[0];
        for (int i = 0; i < rows; i++)
        {
            x = Convert.ToString(i, 2);
            x = x.PadLeft(k, '0');
            characters = x.ToCharArray();
            vector = new bool[characters.Length];
            for (int j = 0; j < characters.Length; j++)
            {
                if (characters[j] == '1')
                {
                    vector[j] = true;
                }
                else
                {
                    vector[j] = false;
                }
            }
            result[i] = new BitArray(vector);
        }
        return result;
    }
    public static List<bool[,]> VectorTable(int k, bool ReturnList)
    {
        int rows = (int)Math.Pow((double)2, (double)k);
        bool[] tmp = new bool[k];
        BitArray[] result = new BitArray[rows];
        string x = string.Empty;
        char[] characters = new char[0];
        bool[] vector = new bool[0];
        for (int i = 0; i < rows; i++)
        {
            x = Convert.ToString(i, 2);
            x = x.PadLeft(k, '0');
            characters = x.ToCharArray();
            vector = new bool[characters.Length];
            for (int j = 0; j < characters.Length; j++)
            {
                if (characters[j] == '1')
                {
                    vector[j] = true;
                }
                else
```

```
                {
                    vector[j] = false;
                }
            }
            result[i] = new BitArray(vector);
        }

        List<bool[,]> output = new List<bool[,]>();

        if (ReturnList)
        {
            for (int i = 0; i < result.Length; i++)
            {
                output.Add(BitsToBoolRow(result[i]));
            }
        }
        return output;
    }


    public static bool[,] BitsToBoolColumn(BitArray bits)
    {
        bool[,] result = new bool[bits.Length, 1];

        for (int i = 0; i < bits.Count; i++)
        {
            result[i, 0] = bits.Get(i);
        }
        return result;
    }
    public static bool[,] BitsToBoolRow(BitArray bits)
    {
        bool[,] result = new bool[1, bits.Length];

        for (int i = 0; i < bits.Count; i++)
        {
            result[0, i] = bits.Get(i);
        }
        return result;
    }

    public static BitArray BoolColumnToBits(bool[,] x)
    {
        BitArray result = new BitArray(x.GetLength(0));

        for (int i = 0; i < x.GetLength(0); i++)
        {
            result.Set(i, x[i, 0]);
        }
        return result;
    }
    public static BitArray BoolRowToBits(bool[,] x)
    {
        BitArray result = new BitArray(x.GetLength(1));

        for (int i = 0; i < x.GetLength(1); i++)
        {
            result.Set(i, x[0, i]);
        }
        return result;
    }
```

```
//сравнение набора битов
public static bool BitArrayEquality(BitArray A, BitArray B)
{
    if (A.Length != B.Length)
    {
        return false;
    }
    int count = 0;
    for (int i = 0; i < A.Length; i++)
    {
        if (A.Get(i) != B.Get(i))
        {
            count++;
        }
    }

    if (count > 0)
    {
        return false;
    }
    else
    {
        return true;
    }

}

//Набор битов в десятичное число
public static int Int(BitArray bitArray)
{

    if (bitArray.Length > 32)
        throw new ArgumentException("Argument length shall be at most 32 bits.");

    int[] array = new int[1];
    bitArray.CopyTo(array, 0);
    return array[0];

}


// умножение матриц
public static bool[,] Multiply_matrix(bool[,] matrix1, bool[,] matrix2)
{
    if (matrix1.GetLength(1) != matrix2.GetLength(0))
        throw new ArgumentException("Invalid arrays length");



    int commonLength = matrix1.GetLength(1);

    bool[,] res = new bool[matrix1.GetLength(0), matrix2.GetLength(1)];

    for (int i = 0; i < res.GetLength(0); i++)
    {
        for (int j = 0; j < res.GetLength(1); j++)
        {
            bool nextVal = false;
            for (int k = 0; k < commonLength; k++)
            {
                nextVal ^= matrix1[i, k] & matrix2[k, j];
```

```
                    }
                    res[i, j] = nextVal;
                }
            }
            return res;
        }

    }
}


//файл Print.cs
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Approximations
{
    class Print
    {

        public static void Bits(BitArray f)
        {
            int count = 0;
            for (int i = 0; i < f.Length; i++)
            {
                if (f[i] == true)
                { Console.Write("1"); }
                else
                { Console.Write("0"); }
                count++;

                if (count == 8)
                {
                    count = 0;
                    Console.Write(" ");
                }

            }
            Console.WriteLine();
        }
        public static void Bits_ext(BitArray f)
        {
            int count = 0;
            for (int i = 0; i < f.Length; i++)
            {
                if (f[i] == true)
                { Console.Write("1"); }
                else
                { Console.Write("0"); }
                count++;

                if (count == 8)
                {
                    count = 0;
                    Console.Write(" ");
                }

            }
```

```
}

public static void Matrix(int[,] res)
{

    for (int i = 0; i < res.GetLength(0); i++)
    {
        for (int j = 0; j < res.GetLength(1); j++)
        {
            Console.Write(res[i, j]);
        }
        Console.WriteLine();
    }
    Console.WriteLine();
}
public static void Matrix(byte[,] res)
{

    for (int i = 0; i < res.GetLength(0); i++)
    {
        for (int j = 0; j < res.GetLength(1); j++)
        {
            Console.Write("{0:x}" + " ", res[i, j]);
        }
        Console.WriteLine();
    }
    Console.WriteLine();
}
public static void Matrix(bool[,] res)
{

    for (int i = 0; i < res.GetLength(0); i++)
    {
        for (int j = 0; j < res.GetLength(1); j++)
        {
            if (res[i, j] == true)
            { Console.Write(1); }
            else
            { Console.Write(0); }
        }
        Console.WriteLine();
    }
    Console.WriteLine();
}


public static void Array(byte[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        Console.Write("{0} ", array[i]);
    }
    Console.WriteLine();
}
public static void Array(bool[] array)
{
    for (int i = 0; i < array.Length; i++)
    {
        if (array[i])
        {
            Console.Write(1);
        }
```

```
                else
                {
                    Console.Write(0);
                }
            }
            Console.WriteLine();
        }
        public static void Array(int[] array)
        {
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write(array[i] + " ");
            }
            Console.WriteLine();
        }
        public static void Array(string[] array)
        {
            for (int i = 0; i < array.Length; i++)
            {
                Console.Write(array[i] + " ");
            }
            Console.WriteLine();
        }

        public static void Array(bool[] array, int size)
        {
            int count = 0;
            for (int i = 0; i < array.Length; i++)
            {
                if (array[i])
                {
                    Console.Write(1);
                }
                else
                {
                    Console.Write(0);
                }
                count++;

                if (count == size)
                {
                    count = 0;
                    Console.WriteLine();
                }
            }

            Console.WriteLine();

        }


        public static void List(List<bool[,]> list)
        {
            for (int i = 0; i < list.Count; i++)
            {
                Print.Matrix(list[i]);
            }

        }
    }
}
```

```
//файл Program.cs
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Runtime.InteropServices;
using System.Collections;
using System.Reflection;
using System.Collections.Specialized;
using Org.BouncyCastle.Math;

namespace Approximations
{
    class Program
    {

        static class MapleEngine
        {
            // interface callback definitions
            public delegate void TextCallBack(IntPtr data, int tag, IntPtr output);
            public delegate void ErrorCallBack(IntPtr data, IntPtr offset, IntPtr msg);
            public delegate void StatusCallBack(IntPtr data, IntPtr used, IntPtr alloc,
double time);
            public delegate IntPtr ReadLineCallBack(IntPtr data, IntPtr debug);
            public delegate long RedirectCallBack(IntPtr data, IntPtr name, IntPtr mode);
            public delegate IntPtr StreamCallBack(IntPtr data, IntPtr stream, int nargs,
IntPtr args);
            public delegate long QueryInterrupt(IntPtr data);
            public delegate IntPtr CallBackCallBack(IntPtr data, IntPtr output);

            public struct MapleCallbacks
            {
                public TextCallBack textCallBack;
                public ErrorCallBack errorCallBack;
                public StatusCallBack statusCallBack;
                public ReadLineCallBack readlineCallBack;
                public RedirectCallBack redirectCallBack;
                public StreamCallBack streamCallBack;
                public QueryInterrupt queryInterrupt;
                public CallBackCallBack callbackCallBack;
            }

            // OpenMaple API methods (there are many more commands in the API,
            // these are just the ones we are using in this example)
            [DllImport(@"maplec.dll")]
            public static extern IntPtr StartMaple(int argc, String[] argv, ref
MapleCallbacks cb, IntPtr data, IntPtr info, byte[] err);
            [DllImport(@"maplec.dll")]
            public static extern IntPtr EvalMapleStatement(IntPtr kv, byte[] statement);
            [DllImport(@"maplec.dll")]
            public static extern IntPtr IsMapleStop(IntPtr kv, IntPtr obj);
            [DllImport(@"maplec.dll")]
            public static extern void StopMaple(IntPtr kv);
        }


        class MainApp
        {
```

```
        public static string Last_maple_output = "";
        // When evaluating something Maple will send all displayed
        // output through this function.
        public static void cbText(IntPtr data, int tag, IntPtr output)
        {
            Last_maple_output = Marshal.PtrToStringAnsi(output);
        }

        // When evaluating something Maple will send errors through this function.
        // If not defined, errors will go through the text callback.
        public static void cbError(IntPtr data, IntPtr offset, IntPtr msg)
        {
            //Console.WriteLine("offset is " + offset.ToInt32() );
            string s = Marshal.PtrToStringAnsi(msg);
            Console.WriteLine(s);
        }

        // When evaluating something Maple will send a message about resources
        // used once per garbage collection.  If you don't want to see these
        // messages, just comment out the WriteLine command inside.
        public static void cbStatus(IntPtr data, IntPtr used, IntPtr alloc, double time)
        {
            /*
            Console.WriteLine("cputime=" + time
              + "; memory used=" + used.ToInt64() + "kB"
              + " alloc=" + alloc.ToInt64() + "kB"
            );
             */
        }
    }
```

```
    private static List<int[,]> Build_A_Matrixes(int n, int k, List<string[]> many_A)
    {
        List<int[,]> Finall_A = new List<int[,]>();
        int[,] A = new int[n, k];

        for (int q = 0; q < many_A.Count; q++)
        {
            A = new int[n, k];
            for (int i = 0; i < A.GetLength(0); i++)
            {
                for (int j = 0; j < A.GetLength(1); j++)
                {
                    A[i, j] = int.Parse(many_A[q][j].Substring(i, 1));
                }
            }
            Finall_A.Add(A);
        }



        for (int i = 0; i < Finall_A.Count; i++)
        {
            if (Rank.Of(Operations.TransposeMatrix_N_K_to_K_N(Finall_A[0])) != k)
            {
```

```
                    Finall_A.Remove(Finall_A[i]);
                }
            }



            return Finall_A;
        }


        private static void Build_Fi_Function(int n, int k, string Carlet_Fang_Function,
List<int[,]> Finall_A, out List<bool[,]> F_table, out string[] Fi_function_vectors)
        {



            //Поиск функции Fi от k переменных

            F_table = Operations.VectorTable(n, true);


            List<bool[,]> Fi_table = Operations.VectorTable(k, true);

            List<bool[,]> Fi_vectors = new List<bool[,]>();
            List<List<bool[,]>> All_vectors = new List<List<bool[,]>>();

            for (int j = 0; j < Finall_A.Count; j++)
            {
                for (int i = 0; i < F_table.Count; i++)
                {
                    Fi_vectors.Add(Operations.Multiply_matrix(F_table[i],
Operations.IntMatrixToBool(Finall_A[j])));
                }

                All_vectors.Add(Fi_vectors);
                Fi_vectors = new List<bool[,]>();
            }



            Fi_function_vectors = new string[All_vectors.Count];
            for (int i = 0; i < All_vectors.Count; i++)
            {
                Fi_function_vectors[i] = Build_Fi_Function_Values_Vector(n, k,
Carlet_Fang_Function, Fi_table, All_vectors[i]);
            }




        }


        private static string Build_Fi_Function_Values_Vector(int n, int k, string
Carlet_Fang_Function, List<bool[,]> Fi_table, List<bool[,]> Fi_vectors)
        {


            string Fi_function_vector;
            //Построение вектора значений функции от k переменных
            Fi_function_vector = "";
            int criteria = (int)Math.Pow((double)2, (double)n - k - 1);
```

```
            int count_ones = 0;

            for (int i = 0; i < Fi_table.Count; i++)
            {
                for (int j = 0; j < Fi_vectors.Count; j++)
                {
                    if (Operations.BitArrayEquality(Operations.BoolRowToBits(Fi_table[i]),
Operations.BoolRowToBits(Fi_vectors[j])))
                    {
                        if (Carlet_Fang_Function[j] == '1')
                        {
                            count_ones++;
                        }
                    }
                }
                if (count_ones >= criteria)
                {
                    Fi_function_vector = Fi_function_vector + "1";
                }
                else
                {
                    Fi_function_vector = Fi_function_vector + "0";
                }
                count_ones = 0;
            }
            return Fi_function_vector;
        }




        public static string[] Build_All_G_Functions(int k, List<bool[,]> F_table, string[]
Fi_function_vectors, List<int[,]> Finall_A, out List<All_G_functions> G_functions_List)
        {
            string[] result = new string[Finall_A.Count];
            for (int i = 0; i<Finall_A.Count; i++)
            {
                result[i] = Build_One_G_Function(k, F_table, Fi_function_vectors[i],
Finall_A[i]);
            }

            G_functions_List = new List<All_G_functions>();
            for (int i = 0; i < Finall_A.Count; i++)
            {
                G_functions_List.Add(new All_G_functions(Finall_A[i],
Fi_function_vectors[i], result[i]));
            }
            return result;
        }

        private static string Build_One_G_Function(int k, List<bool[,]> F_table, string
Fi_function_vector, int[,] Finall_A)
        {
            string G_function = "";

            BitArray EvaluateVector = new BitArray(k);

            for (int i = 0; i < F_table.Count; i++)
```

```
            {
                EvaluateVector =
Operations.BoolRowToBits(Operations.Multiply_matrix(F_table[i],
Operations.IntMatrixToBool(Finall_A)));
                if (Fi_function_vector[Operations.Int(EvaluateVector)] == '1')
                {
                    G_function = G_function + "1";
                }
                else
                {
                    G_function = G_function + "0";
                }
            }
            return G_function;
        }



        public struct All_G_functions
        {
            private int[,] A;
            private string Fi_vector;
            private string G_vector;

            public All_G_functions(int[,] A, string Fi_vector, string G_vector)
            {
                this.A = A;
                this.Fi_vector = Fi_vector;
                this.G_vector = G_vector;

            }

            public void View_G_Function()
            {
                Console.WriteLine("A Matrix:");
                Print.Matrix(A);
                Console.WriteLine("Fi vector");
                Console.WriteLine(Fi_vector);
                Console.WriteLine("G vector");
                Console.WriteLine(G_vector);
                Console.WriteLine();
            }
        }

        private static void Build_Material(int k, string[] vectors, out List <string[]>
many_A)
        {

            many_A = new List<string[]>();
            string[] components_of_A = new string[k];


            List<int[]> counters = Build_Counters(k, vectors);

            for (int i = 0; i < counters.Count; i++)
            {
                for (int j = 0; j < counters[i].Length; j++)
                {
                    components_of_A[j] = vectors[counters[i][j]];
                }
                many_A.Add(components_of_A);
                components_of_A = new string[k];
```

```
            }



        }

        static List<int> tmp = new List<int>();
        static List<int[]> counters = new List<int[]>();

        private static List<int[]> Build_Counters(int k, string[] vectors)
        {

            int[] length = new int[k];

            for (int i = 0; i < length.Length; i++)
            {
                length[i] = vectors.Length;
            }

            int[] counters = new int[k];

            nestedLoopOperation(counters, length, 0);

            // Проверка на повторение элементов в массивах
            List<int[]> tmp = new List<int[]>();

            for (int i = 0; i < Program.counters.Count; i++)
            {
                if (Program.counters[i].Distinct().ToArray().Length == k)
                {

                    tmp.Add(Program.counters[i]);


                }

            }
            ////////////////////////////////////////////////



            // Выборка сортированных элементов
            tmp.RemoveAll(SortedOrNot);



            return tmp;
        }

        static void nestedLoopOperation(int[] counters, int[] length, int level)
        {

            if (level == counters.Length)
            {
                performOperation(counters);
            }
            else
            {
                for (counters[level] = 0; counters[level] < length[level];
counters[level]++)
                {
                    nestedLoopOperation(counters, length, level + 1);
```

```
            }
        }
    }
    static void performOperation(int[] counters)
    {
        tmp = new List<int>();
        for (int i = 0; i<counters.Length; i++)
        {
            tmp.Add(counters[i]);
        }

        Program.counters.Add(tmp.ToArray());
    }
    private static bool SortedOrNot(int[] array)
    {
        bool flag = true;
        for (int i = 1; i < array.Length; i++)
        {
            if (!(array[i - 1] > array[i]))
            {
                flag = flag && true;
            }
            else
            {
                flag = false;
                break;
            }
        }
        return !flag;
    }


    static void Test_Carlet_Feng_k_3()
    {
        #region
        // Число переменных исходной фукнции
        int n = 5;

        // Число переменных фукнции аппроксиманта (Fi)
        int k = 3;

        // Вектор значений исходной фукнции
        string Carlet_Fang_Function = "11111010110110011011001000000010";

        // Таблица векторов для функции от n переменных
        List<bool[,]> F_table;

        // Векторы значений фукнций аппроксимантов (Fi)
        string[] Fi_function_vectors;


        // Векторы значений функций аппроксимантов (G)
        string[] G_functions;

        // Все функции G
        List<All_G_functions> All_G_Functions_List;



        #endregion
```

```
            #region
            // Вектора, полученные после преобразования Уолша-Адамара. См. п.1 алгоритма
            string[] vectors = new string[] {
                                        "10000",
                                        "00101",
                                        "10111",
                                        "11011",
                                        "00001",
                                        "00100",
                                        "01000",
                                        "01001",

                                        "01100",
                                        "10010",
                                        "10011",
                                        "00010",
                                        "01010",
                                        "01011",
                                        "01110",
                                        "10001",

                                        "10101",
                                        "11000",
                                        "11100",
                                        "11101",
                                        "00011",
                                        "00110",
                                        "00111",
                                        "01111",

                                        "10100",
                                        "11001",
                                    };


            #endregion


            List<string[]> many_A;
            //
            Build_Material(k, vectors, out many_A);

            //Построение списка матриц A
            List<int[,]> Finall_A = Build_A_Matrixes(n, k, many_A);

            //Построение функции Fi
            Build_Fi_Function(n, k, Carlet_Fang_Function, Finall_A, out F_table, out
Fi_function_vectors);


            // Поиск функций-аппроксимантов G от n переменных
            G_functions = Build_All_G_Functions(k, F_table, Fi_function_vectors, Finall_A,
out All_G_Functions_List);


            List<G_Function> Functions = new List<G_Function>();
            double[] distances = new double[G_functions.Length];

            for (int i = 0; i < All_G_Functions_List.Count; i++)
            {
                Functions.Add(new G_Function(n, k, Finall_A[i], Fi_function_vectors[i]));
```

```
                distances[i] = Functions[i].Evaluate_Distance(Carlet_Fang_Function);
            }

            Console.WriteLine("Test n = {0}, k = {1}", n, k);

            List<G_Function> Only_Good_Functions = Functions.Where(x =>
x.Evaluate_Distance(Carlet_Fang_Function) == distances.Min()).ToList();

            Console.WriteLine("Input Function:  {0}", Carlet_Fang_Function);

            Console.WriteLine("Approximations found: {0}", Only_Good_Functions.Count);

            Console.WriteLine("Minimal distance: {0}", distances.Min());


            Console.WriteLine();
            Only_Good_Functions.Last().View_G_Function();


        }

        static void Test_Carlet_Feng_k_2()
        {

            #region
            // Число переменных исходной фукнции
            int n = 5;

            // Число переменных фукнции аппроксиманта (Fi)
            int k = 2;

            // Вектор значений исходной фукнции
            string Carlet_Fang_Function = "11111010110110011011001000000010";

            // Таблица векторов для функции от n переменных
            List<bool[,]> F_table;

            // Векторы значений фукнций аппроксимантов (Fi)
            string[] Fi_function_vectors;


            // Векторы значений функций аппроксимантов (G)
            string[] G_functions;

            // Все функции G
            List<All_G_functions> All_G_Functions_List;




            #endregion


            #region
            // Вектора, полученные после преобразования Уолша-Адамара. См. п.1 алгоритма
            string[] vectors = new string[] {
                                            "10000",
                                            "00101",
                                            "10111",
                                            "11011",
                                            "00001",
                                            "00100",
```

```
                                        "01000",
                                        "01001",

                                        "01100",
                                        "10010",
                                        "10011",
                                        "00010",
                                        "01010",
                                        "01011",
                                        "01110",
                                        "10001",

                                        "10101",
                                        "11000",
                                        "11100",
                                        "11101",
                                        "00011",
                                        "00110",
                                        "00111",
                                        "01111",

                                        "10100",
                                        "11001",
                                    };


            #endregion


            List<string[]> many_A;
            //
            Build_Material(k, vectors, out many_A);

            //Построение списка матриц A
            List<int[,]> Finall_A = Build_A_Matrixes(n, k, many_A);


            //Построение функции Fi
            Build_Fi_Function(n, k, Carlet_Fang_Function, Finall_A, out F_table, out
Fi_function_vectors);


            // Поиск функций-аппроксимантов G от n переменных
            G_functions = Build_All_G_Functions(k, F_table, Fi_function_vectors, Finall_A,
out All_G_Functions_List);


            List<G_Function> Functions = new List<G_Function>();
            double[] distances = new double[G_functions.Length];

            for (int i = 0; i < All_G_Functions_List.Count; i++)
            {
                Functions.Add(new G_Function(n, k, Finall_A[i], Fi_function_vectors[i]));
                distances[i] = Functions[i].Evaluate_Distance(Carlet_Fang_Function);
            }



            Console.WriteLine("Test n = {0}, k = {1}", n, k);

            List<G_Function> Only_Good_Functions = Functions.Where(x =>
x.Evaluate_Distance(Carlet_Fang_Function) == distances.Min()).ToList();
```

```
            Console.WriteLine("Input Function:  {0}", Carlet_Fang_Function);

            Console.WriteLine("Approximations found: {0}", Only_Good_Functions.Count);

            Console.WriteLine("Minimal distance: {0}", distances.Min());


            Console.WriteLine();


            Only_Good_Functions.Last().View_G_Function();
        }

        static void Test_Carlet_Feng_k_4()
        {
            #region
            // Число переменных исходной фукнции
            int n = 5;

            // Число переменных фукнции аппроксиманта (Fi)
            int k = 5;

            // Вектор значений исходной фукнции
            string Carlet_Fang_Function = "11111010110110011011001000000010";

            // Таблица векторов для функции от n переменных
            List<bool[,]> F_table;

            // Векторы значений фукнций аппроксимантов (Fi)
            string[] Fi_function_vectors;


            // Векторы значений функций аппроксимантов (G)
            string[] G_functions;

            // Все функции G
            List<All_G_functions> All_G_Functions_List;




            #endregion


            #region
            // Вектора, полученные после преобразования Уолша-Адамара. См. п.1 алгоритма
            string[] vectors = new string[] {
                                              "10000",
                                              "00101",
                                              "10111",
                                              "11011",
                                              "00001",
                                              "00100",
                                              "01000",
                                              "01001",

                                              "01100",
                                              "10010",
                                              "10011",
                                              "00010",
                                              "01010",
```

```
                                        "01011",
                                        "01110",
                                        "10001",

                                        "10101",
                                        "11000",
                                        "11100",
                                        "11101",
                                        "00011",
                                        "00110",
                                        "00111",
                                        "01111",

                                        "10100",
                                        "11001",
                                    };


            #endregion


            List<string[]> many_A;

            Build_Material(k, vectors, out many_A);

            //Построение списка матриц A
            List<int[,]> Finall_A = Build_A_Matrixes(n, k, many_A);



            //Построение функции Fi
            Build_Fi_Function(n, k, Carlet_Fang_Function, Finall_A, out F_table, out
Fi_function_vectors);




            // Поиск функций-аппроксимантов G от n переменных
            G_functions = Build_All_G_Functions(k, F_table, Fi_function_vectors, Finall_A,
out All_G_Functions_List);


            List<G_Function> Functions = new List<G_Function>();
            double[] distances = new double[G_functions.Length];

            for (int i = 0; i < All_G_Functions_List.Count; i++)
            {
                Functions.Add(new G_Function(n, k, Finall_A[i], Fi_function_vectors[i]));
                distances[i] = Functions[i].Evaluate_Distance(Carlet_Fang_Function);
            }

            Console.WriteLine("Test n = {0}, k = {1}", n, k);

            List<G_Function> Only_Good_Functions = Functions.Where(x =>
x.Evaluate_Distance(Carlet_Fang_Function) == distances.Min()).ToList();

            Console.WriteLine("Input Function:  {0}", Carlet_Fang_Function);

            Console.WriteLine("Approximations found: {0}", Only_Good_Functions.Count);

            Console.WriteLine("Minimal distance: {0}", distances.Min());
```

```
            Console.WriteLine();


            Only_Good_Functions.Last().View_G_Function();
        }

        static void Main(string[] args)
        {
            //Запуск оболочки Maple
            #region
            MapleEngine.MapleCallbacks cb;
            byte[] err = new byte[2048];
            IntPtr kv;

            // pass -A2 which sets kernelopts(assertlevel=2) just to show
            // how in this example.  The corresponding argc parameter
            // (the first argument to StartMaple) should then be 2
            // argv[0] should always be filled in with something.
            String[] argv = new String[2];
            argv[0] = "maple";
            argv[1] = "-A2";

            // only a textcallback is really needed here, but error and status
            // callbacks are also provided to show how to handle the IntPtr
            // arguments
            cb.textCallBack = MainApp.cbText;
            cb.errorCallBack = MainApp.cbError;
            cb.statusCallBack = MainApp.cbStatus;
            cb.readlineCallBack = null;
            cb.redirectCallBack = null;
            cb.streamCallBack = null;
            cb.queryInterrupt = null;
            cb.callbackCallBack = null;



            kv = MapleEngine.StartMaple(2, argv, ref cb, IntPtr.Zero, IntPtr.Zero, err);

            // make sure we got a good kernel vector handle back
            if (kv.ToInt64() == 0)
            {
                // Maple didn't start properly.  The "err" parameter will be filled
                // in with the reason why (usually a license error)
                // Note that since we passed in a byte[] array we need to trim
                // the characters past \0 during conversion to string
                Console.WriteLine("Fatal Error, could not start Maple: "
                        + System.Text.Encoding.ASCII.GetString(err, 0, Array.IndexOf(err,
(byte)0))
                    );
            }

            #endregion

            //Test_Carlet_Feng_k_2();
            //Test_Carlet_Feng_k_3();
            Test_Carlet_Feng_k_4();

            MapleEngine.StopMaple(kv);
        }
    }
```

```
}
//файл Rank.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Approximations
{
    class Rank
    {
        // поиск булевого ранга матрицы
        public static int Of(bool[,] inp_Matr)
        {
            bool[,] buffer = new bool[inp_Matr.GetLength(0), inp_Matr.GetLength(1)];
            Array.Copy(inp_Matr, buffer, inp_Matr.GetLength(0) * inp_Matr.GetLength(1));
            int count = 0;
            if (buffer.GetLength(0) > 1 && buffer.GetLength(1) > 1)
            {
                ToOneBool(0, 0, buffer, false);
            }
            else
            {
                int end = buffer.GetLength(0) < buffer.GetLength(1) ? buffer.GetLength(0) :
buffer.GetLength(1);
                if (buffer.GetLength(0) < buffer.GetLength(1))
                {
                    for (int i = 0; i < buffer.GetLength(1); i++)
                    {
                        if (buffer[0, i] != false) { count++; break; }
                    }
                    return count;
                }
                else
                {
                    for (int i = 0; i < buffer.GetLength(0); i++)
                    {
                        if (buffer[i, 0] != false) { count++; break; }
                    }
                    return count;
                }
            }
            for (int i = 0; i < buffer.GetLength(0); i++)
            {
                for (int j = 0; j < buffer.GetLength(1); j++)
                {
                    if (buffer[i, j] != false) { count++; break; }

                }

            }
            return count;
        }
        public static int Of(int[,] inp_Matr)
        {
            bool[,] input = IntMatrixToBool(inp_Matr);

            bool[,] buffer = new bool[inp_Matr.GetLength(0), inp_Matr.GetLength(1)];
            Array.Copy(input, buffer, input.GetLength(0) * input.GetLength(1));
            int count = 0;
            if (buffer.GetLength(0) > 1 && buffer.GetLength(1) > 1)
            {
```

```
                ToOneBool(0, 0, buffer, false);
            }
            else
            {
                int end = buffer.GetLength(0) < buffer.GetLength(1) ? buffer.GetLength(0) :
buffer.GetLength(1);
                if (buffer.GetLength(0) < buffer.GetLength(1))
                {
                    for (int i = 0; i < buffer.GetLength(1); i++)
                    {
                        if (buffer[0, i] != false) { count++; break; }
                    }
                    return count;
                }
                else
                {
                    for (int i = 0; i < buffer.GetLength(0); i++)
                    {
                        if (buffer[i, 0] != false) { count++; break; }
                    }
                    return count;
                }
            }
            for (int i = 0; i < buffer.GetLength(0); i++)
            {
                for (int j = 0; j < buffer.GetLength(1); j++)
                {
                    if (buffer[i, j] != false) { count++; break; }

                }

            }
            return count;
        }

        public static void ToOneBool(int row, int tree, bool[,] arr, bool flag)
        {
            int j = tree;
            /*ищем не нулевой елемент в первом столбце, если этот елемент является
             первым, переходим к шагу 1.1, в противном случае меняем строку с не
             нулевым елементом местами с первой строкой-> к шагу 1.1, если елемент
             не найден, отбрасываем первый столбец*/
            for (int i = row; i < arr.GetLength(0); i++)//1 часть
            {
                if (arr[i, j] != false && flag == false)
                {
                    if (i == row) break;
                    for (int w = tree; w < arr.GetLength(1); w++)
                    {
                        swap(row, w, i, w, arr);
                    }
                    flag = true;
                    break;
                }
                else if (flag == false && tree < arr.GetLength(1) - 1 && i ==
arr.GetLength(0) - 1)
                {
                    ToOneBool(row, tree + 1, arr, false);
                }
            }
            /*если 'элемент проверяемого столбца ненулевой - складываем по модулю 2 первую
строку и строку с ненулевым элементом */
```

```
            for (int i = row + 1; i < arr.GetLength(0); i++)
            {//2 часть
                if (arr[i, tree] == true)
                {
                    for (int tr = tree; tr < arr.GetLength(1); tr++)
                    {
                        arr[i, tr] = arr[i, tr] ^ arr[row, tr];
                    }
                }
            }
            /*Отбрасываем первый столбец и первую строку(следовательно вторая строка
становится первой по счету и второй столбец тоже) -> к шагу 1*/
            if (row < arr.GetLength(0) - 1 && tree < arr.GetLength(1) - 1)
            {
                ToOneBool(row + 1, tree + 1, arr, false);
            }

        }

        private static void swap(int x1, int y1, int x2, int y2, bool[,] arr)
        {
            arr[x1, y1] ^= arr[x2, y2];
            arr[x2, y2] = arr[x1, y1] ^ arr[x2, y2];
            arr[x1, y1] ^= arr[x2, y2];
        }



        public static bool[,] IntMatrixToBool(int[,] A)
        {
            bool[,] result = new bool[A.GetLength(0), A.GetLength(1)];
            for (int i = 0; i < A.GetLength(0); i++)
            {
                for (int j = 0; j < A.GetLength(1); j++)
                {
                    if (A[i, j] == 1)
                    {
                        result[i, j] = true;
                    }
                    else
                    {
                        result[i, j] = false;
                    }
                }
            }
            return result;
        }
    }
}
```

Додаток В

Программний код реалізації методу обґрунтування відсутності високоймовірних наближень булевих функцій

Програмна реалізація методу обґрунтування відсутності високоймовірних наближень булевих функцій виконана на ПК з процесором Intel Core i7 (1,6 ГГц) та обсягом оперативної пам’яті 4 Гб RAM (DDR3) на базі Windows 7 (використовувався пакет прикладних програм Maple 13.0).

```
restart:

randomize():
with(LinearAlgebra):
with(Bits):

Hadamar := proc (M1, d)
global ResultVector:

if d = 1
then
    ResultVector := [M1[1]+M1[2], M1[1]-M1[2]]:
else
    ResultVector := Vector([Hadamar(SubVector(M1,[1..2^(d-1)])
+SubVector(M1,[2^(d-1)+1..2^d]),d-1),
Hadamar(SubVector(M1,[1..2^(d-1)])-SubVector(M1,[2^(d-
1)+1..2^d]),d-1)]):
end if:
end proc:

Oracle := proc (x)
local x1, x2, g, h1, h2, nx2, mm:

mm := (2^45 + 5)/3:
x1 := SubVector(x, [1 .. 20]):
x2 := SubVector(x, [21 .. 64]):
g := (x1[1]*Multiply(x1, Vector[column](1..20,1)),2):
nx2 := Multiply(x2,Vector[column](1..44,f0)):

if nx2 < (2^45-3*mm)
     then
          h1 := 1:
     else
          h1 := 0:
```

```
end if:

if nx2 < mm
     then
          h2 := 1:
     else
          h2 := 0:
end if:
'mod'(g*h1+(g+1)*h2, 2):
end proc:


n := 64:
k := 2:
m := (2^44+5)*(1/3):
ee := .5:
dd := 0.0625:



fd := fopen("D:/Docs/4.2/temp_file.txt", WRITE):
RANDBYTES := readbytes("D:/Docs/4.2/rand.txt", 10000000):

for iter to 10 do
print(iter):
randomize():
tmG := time():

cc := ceil(log[2](1/(ee^2*dd))):
t := k+cc:
vf := Vector(2^t):
ByH := Vector[column](2^k):

f := proc (j)
     options operator, arrow; 2^(t-j)
end proc;

fO := proc (j)
     options operator, arrow; 2^(44-j)
end proc;

A := Matrix(t, n):
for ro to t do
     for co to n do
          A[ro,
co]:=Split(RANDBYTES[RandomTools[Generate](integer(range=1..100000
00))],bits=8)[RandomTools[Generate](integer(range = 1 .. 8))]:
     end do:
end do:
writedata(fd, convert(Transpose(A), matrix), float):
```

```
for i from 0 to 2^t-1 do

vv[i]:=convert(ListTools[Reverse](Split(i,bits=t)),Vector[row]):
     vf[i+1]:=1-2*Oracle(`mod`(VectorMatrixMultiply(vv[i],A),2)):
end do;
vf[1] := 0:

writedata(fd, convert(Transpose(vf), matrix), float):
vH := Hadamar(vf,t):
writedata(fd, convert(Transpose(vH), matrix), float):
maxN := -1:
for i1 to t-2 do
     B := Matrix(t, k):

     for i2 from i1+1 to i1+1 do
          B := Matrix(t, k):
          B[i1, 1] := 1:
          B[i2, 2] := 1:
          lv := t-i2:

          for dv2 from 0 to 2^lv-1 do
               B[i2+1..t,1]:=convert(ListTools[Reverse](Split(dv2,
bits = lv)),Vector[column]):

               for dv3 from 0 to 2^lv-1 do
               B[i2+1 .. t, 2] :=
convert(ListTools[Reverse](Split(dv3, bits = lv)),
Vector[column]):

                    for y to 2^k-1 do
                         ByH[y+1]:=
vH[Multiply(Vector[row](1..t,f),`mod`(MatrixVectorMultiply(B,
convert(ListTools[Reverse](Split(y,bits=k)),Vector[column])),2))+1
]:
                    end do;
                    ByH[1] := 0: BH := Hadamar(ByH, k):
                    Ny := Multiply(Vector[row](1 .. 2^k, 1),
abs(BH)):
                    writedata(fd, convert(B, matrix), float):
                    writedata(fd, convert(Transpose(BH), matrix),
float):
                    fprintf(fd, "%g\n\n", Ny):
                         if Ny > maxN
                              then
                                   maxN := Ny
                         end if
               end do:
          end do:
     end do;

     for i2 from i1+2 to t-1 do
```

```
          B := Matrix(t, k):
          B[i1, 1] := 1:
          B[i2, 2] := 1:
          lv1 := i2-i1-1:

          for dv1 from 0 to 2^lv1-1 do
               B[i1+1 .. i2-1, 1] :=
convert(ListTools[Reverse](Split(dv1, bits =
lv1)),Vector[column]):
lv := t-i2:

               for dv2 from 0 to 2^lv-1 do
                    B[i2+1..t,1]:=
convert(ListTools[Reverse](Split(dv2, bits = lv)),Vector[column]):

                    for dv3 from 0 to 2^lv-1 do
                         B[i2+1..t,2] :=
convert(ListTools[Reverse](Split(dv3, bits = lv)),Vector[column]):
ByH[1]:=0:

                         for y to 2^k-1 do
                              ByH[y+1]:=
vH[Multiply(Vector[row](1..t,f),

`mod`(MatrixVectorMultiply(B,convert(ListTools[Reverse](Split(y,bi
ts = k)),
Vector[column])), 2))+1]:
                         end do:

                         BH := Hadamar(ByH, k):
                         Ny := Multiply(Vector[row](1 .. 2^k, 1),
abs(BH)):
                         writedata(fd, convert(B, matrix), float):
                         writedata(fd, convert(Transpose(BH),
matrix), float):
                         fprintf(fd, "%g\n\n", Ny):
                              if Ny > maxN
                                   then
                                        maxN:=Ny
                              end if:
                    end do:
               end do:
          end do:
     end do:
     for i2 from t to t do
          B:=Matrix(t, k):
          B[i1, 1]:=1:
          B[i2, 2]:= 1:
          lv1:= i2-i1-1:

          for dv1 from 0 to 2^lv1-1 do
```

```
                B[i1+1..i2-
1,1]:=convert(ListTools[Reverse](Split(dv1, bits =
lv1)),Vector[column]):
ByH[1] := 0:
                for y to 2^k-1 do

ByH[y+1]:=vH[Multiply(Vector[row](1..t,f),`mod`(MatrixVectorMultip
ly(B,
convert(ListTools[Reverse](Split(y, bits = k)),
                     Vector[column])), 2))+1]:
                end do;
                BH := Hadamar(ByH, k):
                Ny := Multiply(Vector[row](1 .. 2^k, 1), abs(BH)):
                writedata(fd, convert(B, matrix), float):
                writedata(fd, convert(Transpose(BH), matrix),
float):
                fprintf(fd, "%g\n\n", Ny):
                     if Ny > maxN
                          then
                               maxN := Ny:
                     end if:
                end do:
           end do:
           B := Matrix(t, k):
           B[t-1, 1] := 1:
           B[t, 2] := 1:
           ByH[1] := 0:
                for y to 2^k-1 do
                     ByH[y+1] := vH[Multiply(Vector[row](1 .. t,
f), `mod`(MatrixVectorMultiply(B,
convert(ListTools[Reverse](Split(y, bits = k)),
                     Vector[column])), 2))+1]
                end do:

           BH := Hadamar(ByH, k):
           Ny := Multiply(Vector[row](1 .. 2^k, 1), abs(BH)):
           writedata(fd, convert(B, matrix), float):
           writedata(fd, convert(Transpose(BH), matrix), float):
           fprintf(fd, "%g\n\n", Ny):
                if Ny > maxN
                     then
                          maxN := Ny:
                end if:
     end do:
     maxN := maxN/(2^k*(2^t-1)):
     print(%);

     teta := evalf(maxN):
     print(%);

     .5*(1-teta-ee):
```

```
      print(%);

      -.25+3*m/2^45:
      print(%);

      time()-tmG:
      print(%);
end do:
fclose(fd):
```

Додаток Д

Программний код реалізації методу пошуку алгебраїчно вироджених наближень булевих функцій для оцінювання стійкості синхронних потокових шифрів відносно узагальненої статистичної атаки

Програмна реалізація методу пошуку алгебраїчно вироджених наближень булевих функцій для оцінювання стійкості синхронних потокових шифрів відносно узагальненої статистичної атаки виконана на ПК з процесором Intel(R) Core(TM) i5-2300 CPU @ 2.80GHz та обсягом оперативної пам’яті 4 Гб RAM на базі 32-розрядної ОС Windows 7 Service Pack 1. Мова програмування – C# .NET Framework 4.0. Середовище розробки – Microsoft Visual Studio 2010.

```
//файл AlgorithmOne.cs
using System;
using System.Linq;
using System.Collections;
using System.Security.Cryptography;

namespace ApproximationsSearchFramework
{
     class AlgorithmOne
     {
          //Calculation accuracy
          double epsilon;

          //Error probability
          double delta;

          //How many times to calculate result
          int times;
          private IOracle oracle;

          public AlgorithmOne(IOracle oracle)
          {
               // TODO: Complete member initialization
               this.oracle = oracle;
               epsilon = 0.0;
               delta = 0.0;
               times = 0;
          }

          internal void Perform()
          {
               Console.WriteLine("STEP 1");
               Console.WriteLine("Calculating the value of the Mu^2 parameter upper
bound...");
```

```
                SelectParameters();

                //User pressed 'q' while selecting epsilon/delta
                if (epsilon == 0.0 || delta == 0.0 || times == 0)
                {
                    Console.WriteLine("Bye");
                    return;
                }


                Console.WriteLine("Calculating . . .");
                CalculateMuSquared muSquaredValue = new CalculateMuSquared(oracle, epsilon,
delta);
                var results = muSquaredValue.CalculateResults(times);

                PrintResults(results);

                Console.WriteLine();
                Console.WriteLine("------------------------------------------------------------
");

            }

            private void PrintResults(AlgorithmOneResults results)
            {
                Console.Write("Do you want to see all results (if no, only average will be
shown) ? (y/n) ");

                string userSelect = Console.ReadLine();
                Console.WriteLine();

                Console.WriteLine("Algorithm 1 results (value of Mu^2 parameter upper
bound)");
                Console.WriteLine("ORACLE - {0}, EPSILON = {1}, DELTA = {2} ",
oracle.ToString(), epsilon, delta);
                Console.WriteLine();

                if (userSelect == "y")
                {
                    for (int resultNumber = 0; resultNumber < times; resultNumber++)
                    {
                        Console.WriteLine(results.AllResults[resultNumber]);
                    }
                    Console.WriteLine();
                }


                Console.ForegroundColor = ConsoleColor.Green;
                Console.WriteLine("Average (MU-squared) is {0}, MU: {1}", results.Average,
Math.Sqrt(results.Average));
                Console.ResetColor();

            }

            private void SelectParameters()
            {

            Selecting_epsilon:

                Console.WriteLine();
                Console.WriteLine("Select EPSILON:");
                Console.WriteLine("1 - 0,01");
```

```
            Console.WriteLine("2 - 0,05");
            Console.WriteLine("3 - 0,1");
            Console.WriteLine("4 - 0,25");

            string userSelect = Console.ReadLine();
            Console.WriteLine();

            switch (userSelect)
            {
                  case "1": epsilon = 0.01; break;
                  case "2": epsilon = 0.05; break;
                  case "3": epsilon = 0.1; break;
                  case "4": epsilon = 0.25; break;

                  case "q": return;

                  default:
                        Console.WriteLine("Please select only 1,2,3 or 4");
                        goto Selecting_epsilon;
            }



      Selecting_delta:

            Console.WriteLine("Select DELTA:");
            Console.WriteLine("1 - 0,01");
            Console.WriteLine("2 - 0,05");
            Console.WriteLine("3 - 0,1");
            Console.WriteLine("4 - 0,25");

            userSelect = Console.ReadLine();
            Console.WriteLine();

            switch (userSelect)
            {
                  case "1": delta = 0.01; break;
                  case "2": delta = 0.05; break;
                  case "3": delta = 0.1; break;
                  case "4": delta = 0.25; break;

                  case "q": return;

                  default:
                        Console.WriteLine("Please select only 1,2,3 or 4");
                        goto Selecting_delta;
            }

      Selecting_times:

            Console.WriteLine("How many times to calculate:");
            Console.WriteLine("1 - 1");
            Console.WriteLine("2 - 5");
            Console.WriteLine("3 - 10");
            Console.WriteLine("4 - 20");

            userSelect = Console.ReadLine();
            Console.WriteLine();

            switch (userSelect)
            {
                  case "1": times = 1; break;
```

```
                    case "2": times = 5; break;
                    case "3": times = 10; break;
                    case "4": times = 20; break;

                    case "q": return;

                    default:
                         Console.WriteLine("Please select only 1,2,3 or 4");
                         goto Selecting_times;
               }



          }
     }

     internal class AlgorithmOneResults
     {
          private double[] results;
          public double[] AllResults { get { return results; } }

          private double average;
          public double Average { get { return average; } }


          private AlgorithmOneResults()
          { }

          public AlgorithmOneResults(double[] results)
          {
               this.results = results;
               average = results.Average();


          }
     }
}



//файл AlgorithmTwo.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ApproximationsSearchFramework.Algorithms
{
     class AlgorithmTwo
     {
          private IOracle oracle;


          private double epsilon;
          private double delta;
          private int vectorsAmount;

          private int s;
          double mu;
          double theta;
```

```
        public AlgorithmTwo(IOracle oracle)
        {
            this.oracle = oracle;
            this.epsilon = 0;
            this.delta = 0;

            this.vectorsAmount = 0;
            this.s = 0;
            this.mu = 0;
            this.theta = 0;
        }


        internal void Perform()
        {
            Console.WriteLine("STEP 2");
            Console.WriteLine("Searching for linear-independent low-weight vectors");

            SelectParameters();

            //User pressed 'q' while selecting epsilon/delta
            if (epsilon == 0.0 || delta == 0.0 || vectorsAmount == 0)
            {
                Console.WriteLine("Bye");
                return;
            }

            Console.WriteLine("ORACLE - {0}, EPSILON = {1}, DELTA = {2}, S = {3}, MU =
{4}, THETA = {5}, Vectors Amound = {6}", oracle.ToString(), epsilon, delta, s, mu, theta,
vectorsAmount);
            Console.WriteLine("This is a most time-consuming step. If vector is found, it
is highlighted in green and saved to file");
            Console.WriteLine("There are also a vectors printed each 100000 iterated
candidates to be Alpha-vector");

            AlphaVectorsSearching alphaVectors = new AlphaVectorsSearching(oracle,
epsilon, delta, s, theta, mu);
            alphaVectors.Presearch();

            Console.WriteLine("Service parameters: t = {0}; l = {1}; w = {2}",
alphaVectors.T, alphaVectors.L, alphaVectors.W);
            Console.WriteLine();
            Console.WriteLine("Searching . . .");
            List<KeyValuePair<double, byte[]>> foundVectors = new
List<KeyValuePair<double, byte[]>>();


            for (int i = 0; i < vectorsAmount; i++)
            {
                foundVectors.Add(alphaVectors.Search());

                //Print if found
                if (foundVectors.Last().Key != -1)
                {
                    Console.WriteLine();
                    Console.ForegroundColor = ConsoleColor.Green;
                    Console.WriteLine("FOUND LOW_WEIGHT VECTOR");
                    Console.Write("{0}\t|\t", foundVectors.Last().Key);

                    for (int byteIndex = 0; byteIndex <
foundVectors.Last().Value.Length; byteIndex++)
```

```
                    {
                        Console.Write("{0:x}", foundVectors.Last().Value[byteIndex]);
                    }

                    Console.WriteLine();
                    Console.ResetColor();
                }
                else
                {
                    foundVectors.Remove(foundVectors.Last());
                    break;
                }

            }

            if (foundVectors.Count == 0)
            {
                Console.ForegroundColor = ConsoleColor.Green;
                Console.WriteLine("No vectors found");
                Console.ResetColor();
                return;
            }

            Console.WriteLine();
            SortResults(foundVectors);

        }

        private void SortResults(List<KeyValuePair<double, byte[]>> foundVectors)
        {
            Console.WriteLine();
            Console.Write("Do you want to sort result by weight ascendancy? (y/n) ");

            string userSelect = Console.ReadLine();

            Console.WriteLine();

            if (userSelect == "y")
            {
                foundVectors.Sort((x, y) => x.Key.CompareTo(y.Key));
            }

            else
            {
                return;
            }


            for (int foundVectorIndex = 0; foundVectorIndex < foundVectors.Count;
foundVectorIndex++)
            {
                Console.Write("{0}\t|\t", foundVectors[foundVectorIndex].Key);
                for (int byteIndex = 0; byteIndex <
foundVectors[foundVectorIndex].Value.Length; byteIndex++)
                {
                    Console.Write("{0:x} ",
foundVectors[foundVectorIndex].Value[byteIndex]);
                }
                Console.WriteLine();
            }

        }
```

```
private void SelectParameters()
{

Selecting_epsilon:

     Console.WriteLine();
     Console.WriteLine("Select EPSILON:");
     Console.WriteLine("1 - 0,005");
     Console.WriteLine("2 - 0,01");
     Console.WriteLine("3 - 0,05");
     Console.WriteLine("4 - 0,1");

     string userSelect = Console.ReadLine();
     Console.WriteLine();

     switch (userSelect)
     {
          case "1": epsilon = 0.005; break;
          case "2": epsilon = 0.01; break;
          case "3": epsilon = 0.05; break;
          case "4": epsilon = 0.1; break;

          case "q": return;

          default:
               Console.WriteLine("Please select only 1,2,3 or 4");
               goto Selecting_epsilon;
     }


Selecting_delta:

     Console.WriteLine("Select DELTA:");
     Console.WriteLine("1 - 0,01");
     Console.WriteLine("2 - 0,05");
     Console.WriteLine("3 - 0,1");
     Console.WriteLine("4 - 0,25");

     userSelect = Console.ReadLine();
     Console.WriteLine();

     switch (userSelect)
     {
          case "1": delta = 0.01; break;
          case "2": delta = 0.05; break;
          case "3": delta = 0.1; break;
          case "4": delta = 0.25; break;

          case "q": return;

          default:
               Console.WriteLine("Please select only 1,2,3 or 4");
               goto Selecting_delta;
     }


Selecting_s:

     Console.WriteLine("Select S:");
```

```
            Console.WriteLine("1 - 20");
            Console.WriteLine("2 - 30");
            Console.WriteLine("3 - 40");
            Console.WriteLine("4 - 50");

            userSelect = Console.ReadLine();
            Console.WriteLine();

            switch (userSelect)
            {
                case "1": s = 20; break;
                case "2": s = 30; break;
                case "3": s = 40; break;
                case "4": s = 50; break;

                case "q": return;

                default:
                    Console.WriteLine("Please select only 1,2,3 or 4");
                    goto Selecting_s;
            }

        Selecting_theta:

            Console.WriteLine("Select THETA:");
            Console.WriteLine("1 - 0,30");
            Console.WriteLine("2 - 0,40");
            Console.WriteLine("3 - 0,60");
            Console.WriteLine("4 - 0,70");

            userSelect = Console.ReadLine();
            Console.WriteLine();

            switch (userSelect)
            {
                case "1": theta = 0.30; break;
                case "2": theta = 0.40; break;
                case "3": theta = 0.60; break;
                case "4": theta = 0.70; break;

                case "q": return;

                default:
                    Console.WriteLine("Please select only 1,2,3 or 4");
                    goto Selecting_theta;
            }


        Selecting_mu:

            Console.WriteLine("Select MU:");
            Console.WriteLine("1 - 0,50");
            Console.WriteLine("2 - 0,60");
            Console.WriteLine("3 - 0,70");
            Console.WriteLine("4 - 0,80");

            userSelect = Console.ReadLine();
            Console.WriteLine();

            switch (userSelect)
            {
                case "1": mu = 0.50; break;
```

```
                    case "2": mu = 0.60; break;
                    case "3": mu = 0.70; break;
                    case "4": mu = 0.80; break;

                    case "q": return;

                    default:
                        Console.WriteLine("Please select only 1,2,3 or 4");
                        goto Selecting_mu;
                }







            Selecting_amount:

                Console.WriteLine("How many vectors needed?");
                Console.WriteLine("1 - 1");
                Console.WriteLine("2 - 10");
                Console.WriteLine("3 - 50");
                Console.WriteLine("4 - 100");
                Console.WriteLine("5 - 100");

                userSelect = Console.ReadLine();
                Console.WriteLine();

                switch (userSelect)
                {
                    case "1": vectorsAmount = 1; break;
                    case "2": vectorsAmount = 10; break;
                    case "3": vectorsAmount = 50; break;
                    case "4": vectorsAmount = 100; break;
                    case "5": vectorsAmount = 118; break;

                    case "q": return;

                    default:
                        Console.WriteLine("Please select only 1,2,3 or 4");
                        goto Selecting_amount;
                }



            }

        }
    }

    //файл AlphaVectorsSearching.cs
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.IO;

    using System.Numerics;

    namespace ApproximationsSearchFramework.Algorithms
    {
        class AlphaVectorsSearching
```

```
    {
        private IOracle oracle;
        private double epsilon;
        private double delta;
        private int s;
        private double theta;
        private double mu;

        //Alpha vector' candidates to check
        private long t;
        public long T { get {return t;} }


        //Material needed for 1 vector alpha evaluation
        private int l;
        public int L { get { return l; } }


        //Threshold
        double w;
        public double W { get { return w; } }




        public AlphaVectorsSearching(IOracle oracle, double epsilon, double delta, int s,
double theta, double mu)
        {
            if (oracle == null || epsilon == 0.0 || delta == 0 || s == 0 || theta == 0.0
|| mu == 0.0)
            {
                throw new ArgumentException("One of input parameters is not specified");
            }

            // TODO: Complete member initialization
            this.oracle = oracle;
            this.epsilon = epsilon;
            this.delta = delta;
            this.s = s;
            this.theta = theta;
            this.mu = mu;

            t = 0;
            l = 0;
            w = 0;
        }



        internal void Presearch()
        {


            t = (long)Math.Ceiling(Math.Log(2 / delta) * Math.Pow(2, s + 3) / 3);

            l = (int)Math.Ceiling(Math.Pow(epsilon, -2) * Math.Log(Math.Pow(2, -s + 1) *
t * 1 / delta) / 2);
```

```
            w = Math.Min(1 - theta, (0.5 * (1 - Math.Pow(theta, 2))) + mu);

            if (w > 0.5)
            {
                w = 1 - w;
            }
        }



        internal KeyValuePair<double, byte[]> Search()
        {
            if (t == 0 || l == 0 || w == 0)
            {
                throw new ArgumentException("Call Presearch() to specify service
parameters");
            }

            double weight = 0;

            byte[] alphaVectorCandidate = new byte[oracle.l0 / 8];
            byte[] gamma = new byte[1];
            byte[] keyConcatenatedWithIV = new byte[(oracle.l0 + oracle.l1) / 8];


            oracle.Init();

            for (long alphaVectorIndex = 0; alphaVectorIndex < t; alphaVectorIndex++)
            {

                ApproximationsSearchMethod.randomSource.GetBytes(alphaVectorCandidate);

                //alphaVectorCandidate[1] = 0x01;
                //alphaVectorCandidate[3] = 0x01;
                //alphaVectorCandidate[5] = 0x01;
                //alphaVectorCandidate[6] = 0x01;
                //alphaVectorCandidate[12] = 0x01;
                //alphaVectorCandidate[14] = 0x01;
                //alphaVectorCandidate[15] = 0x01;

                //Snow-2.0 results - 1 round. Strong byte' numbers
                alphaVectorCandidate[0] = 0x00;
                alphaVectorCandidate[2] = 0x00;
                alphaVectorCandidate[4] = 0x00;
                alphaVectorCandidate[7] = 0x00;
                alphaVectorCandidate[8] = 0x00;
                alphaVectorCandidate[9] = 0x00;
                alphaVectorCandidate[10] = 0x00;
                alphaVectorCandidate[11] = 0x00;
                alphaVectorCandidate[13] = 0x00;
                //weak byte' numbers
                //[1]; [3]; [5]; [6]; [12]; [14]; [15] - all byte


                //Snow-2.0 results - 2 rounds
                //weak byte' numbers
                //[0]; [1]; [2] - all byte


                //Snow-2.0 results - 3 rounds
                //weak bits
                //derivationVector[12] - only 0x01;
```

```
                    //derivationVector[14] - all byte;



                    weight = EvaluateWeight(gamma, keyConcatenatedWithIV,
alphaVectorCandidate, l);

                    //if (weight > 0.5)
                    //{
                    //     weight -= epsilon;
                    //}
                    //else
                    //{
                    //     weight += epsilon;
                    //}

                    if (alphaVectorIndex % 100000 == 0 & alphaVectorIndex != 0)
                    {
                         //Each 100000 vector will be printed. This is like a progress bar,
to make sure the program works
                         Console.Write(String.Format("{0:0.00000000}", weight) + "\t|\t");
                         for (int i = 0; i < alphaVectorCandidate.Length; i++)
                         {
                              Console.Write("{0:x}", alphaVectorCandidate[i]);
                         }
                         Console.WriteLine();
                    }

                    //If vector found, so interrupt loop and return found vector with it's
weight
                    if (weight <= w - epsilon /*|| weight >= 1 - (w - epsilon)*/)
                    {
                         return new KeyValuePair<double, byte[]>(weight,
alphaVectorCandidate);
                    }

               }

               return new KeyValuePair<double, byte[]>(-1.0, alphaVectorCandidate);

          }

          private double EvaluateWeight(byte[] gamma, byte[] keyConcatenatedWithIV, byte[]
derivationVector, long l)
          {
               double counter = 0;

               bool oracleBit = false;

               for (long i = 0; i < l; i++)
               {
                    ApproximationsSearchMethod.randomSource.GetBytes(keyConcatenatedWithIV);

                    oracleBit = oracle.GetKeystreamBit(keyConcatenatedWithIV);

                    if (oracleBit != TestDerivationFunction(keyConcatenatedWithIV,
derivationVector))
                    {
                         counter++;
                    }
```

```
                }

                return counter / (double)l;
            }


            private bool TestDerivationFunction(byte[] keyConcatenatedWithIV, byte[]
derivationVector)
            {
                for (int keyByteIndex = 0; keyByteIndex < oracle.l0 / 8; keyByteIndex++)
                {
                    keyConcatenatedWithIV[keyByteIndex] ^= derivationVector[keyByteIndex];
                }
                return oracle.GetKeystreamBit(keyConcatenatedWithIV);
            }
        }
    }

    //файл CalculateMuSquared.cs
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    //файл CalculateMuSquared.cs
    namespace ApproximationsSearchFramework
    {
        class CalculateMuSquared
        {
            private double epsilon;
            private double delta;

            IOracle oracle;

            int n;

            byte[] xorResult;

            public CalculateMuSquared(IOracle oracle, double epsilon, double delta)
            {
                if (epsilon == 0.0 || delta == 0.0)
                {
                    throw new NullReferenceException("EPSILON or DELTA is not specified");
                }

                // TODO: Complete member initialization
                this.epsilon = epsilon;
                this.delta = delta;
                this.oracle = oracle;

                n = oracle.l0 + oracle.l1;

                xorResult = new byte[n / 8];
            }



            internal AlgorithmOneResults CalculateResults(int times)
            {
                //Step 1
                int t = (int)Math.Ceiling(2 / epsilon / epsilon * Math.Log(1 / delta));
                ////////////////////////////////////////////////////////////////////////
```

```
            double[] results = new double[times];

            for (int i = 0; i < times; i++)
            {
                results[i] = CalculateOneResult(t);
            }

            return new AlgorithmOneResults(results);
        }


        //Step 2
        private double CalculateOneResult(int t)
        {
            byte[] x = new byte[n / 8];


            byte[] y = new byte[n / 8];


            byte[] z = new byte[n / 8];

            double muSquared = 0.0;


            oracle.Init();

            for (int i = 0; i < t; i++)
            {
                ApproximationsSearchMethod.randomSource.GetBytes(x);
                ApproximationsSearchMethod.randomSource.GetBytes(y);
                ApproximationsSearchMethod.randomSource.GetBytes(z);


                if (oracle.GetKeystreamBit(x) ^ oracle.GetKeystreamBit(Xor(x, y)) ^
oracle.GetKeystreamBit(Xor(x, z)) ^ oracle.GetKeystreamBit(Xor(x, y, z)))
                {
                    muSquared++;
                }

            }

            return muSquared / t + epsilon;
        }




        #region Service methods

        private byte[] Xor(byte[] x, byte[] y)
        {
            for (int i = 0; i < n / 8; i++)
            {
                xorResult[i] = (byte)(x[i] ^ y[i]);
            }

            return xorResult;
        }
```

```
            private byte[] Xor(byte[] x, byte[] y, byte[] z)
            {
                for (int i = 0; i < n / 8; i++)
                {
                    xorResult[i] = (byte)(x[i] ^ y[i] ^ z[i]);
                }

                return xorResult;
            }

            #endregion
        }
    }


    //файл Grain128.cs
    using System;
    using ApproximationsSearchFramework.Oracles.Additional;

    namespace ApproximationsSearchFramework.Oracles
    {
        class Grain128
        {
            private JIBitArray NFSR;
            public byte[] NFS
            {
                get { return NFSR.GetBytes(); }
            }

            private JIBitArray LFSR;
            public byte[] LFS
            {
                get { return LFSR.GetBytes(); }
            }

            const int keysize = 128;

            const int ivsize = 96;


            public Grain128()
            {
                NFSR = new JIBitArray(keysize, false);
                LFSR = new JIBitArray(keysize, false);
            }


            public void Init(JIBitArray key, JIBitArray iv, int rounds)
            {
                /* load registers */
                //fill NFSR bits with key bits
                for (int i = 0; i < keysize; i++)
                {
                    NFSR.Set(i, key.Get(i));
                }


                for (int i = 0; i < ivsize; i++)
                {
                    LFSR.Set(i, iv.Get(i));
                }
```

```
                //96 < i < 128 bits of LFSR - filling with 1
                for (int i = ivsize; i < keysize; i++)
                {
                    LFSR.Set(i, true);
                }



                /* do initial clockings */

                bool outbit;
                bool Lbit;
                bool Nbit;
                for (int i = 0; i < rounds; ++i)
                {
                    outbit = grain_keystream();

                    Lbit = LFSR.Get(127);
                    Nbit = NFSR.Get(127);

                    LFSR.Set(127, outbit ^ Lbit);
                    NFSR.Set(127, outbit ^ Nbit);
                }


            }


            private bool grain_keystream()
            {

                bool outbit = NFSR.Get(2) ^ NFSR.Get(15) ^ NFSR.Get(36) ^ NFSR.Get(45) ^
NFSR.Get(64) ^ NFSR.Get(73) ^ NFSR.Get(89) ^ LFSR.Get(93) ^ (NFSR.Get(12) & LFSR.Get(8)) ^
(LFSR.Get(13) & LFSR.Get(20)) ^ (NFSR.Get(95) & LFSR.Get(42)) ^ (LFSR.Get(60) &
LFSR.Get(79)) ^ (NFSR.Get(12) & NFSR.Get(95) & LFSR.Get(95));
                bool Nbit = LFSR.Get(0) ^ NFSR.Get(0) ^ NFSR.Get(26) ^ NFSR.Get(56) ^
NFSR.Get(91) ^ NFSR.Get(96) ^ (NFSR.Get(3) & NFSR.Get(67)) ^ (NFSR.Get(11) & NFSR.Get(13)) ^
(NFSR.Get(17) & NFSR.Get(18)) ^ (NFSR.Get(27) & NFSR.Get(59)) ^ (NFSR.Get(40) &
NFSR.Get(48)) ^ (NFSR.Get(61) & NFSR.Get(65)) ^ (NFSR.Get(68) & NFSR.Get(84));
                bool Lbit = LFSR.Get(0) ^ LFSR.Get(7) ^ LFSR.Get(38) ^ LFSR.Get(70) ^
LFSR.Get(81) ^ LFSR.Get(96);


                //NFSR = NFSR.ShiftLeft(1);
                //LFSR = LFSR.ShiftLeft(1);
                NFSR.ShiftLeft(1);
                LFSR.ShiftLeft(1);


                NFSR.Set(keysize - 1, Nbit);
                LFSR.Set(keysize - 1, Lbit);


                return outbit;

            }


            public JIBitArray Generate_Gamma_Bits(int length_in_bits)
            {
                JIBitArray ret = new JIBitArray(length_in_bits);
```

```
                for (int i = 0; i < length_in_bits; i++)
                {
                     ret.Set(i, grain_keystream());
                }
                return ret;

           }


           public bool Generate_Gamma_One_Bit()
           {
                return NFSR.Get(2) ^ NFSR.Get(15) ^ NFSR.Get(36) ^ NFSR.Get(45) ^
NFSR.Get(64) ^ NFSR.Get(73) ^ NFSR.Get(89) ^ LFSR.Get(93) ^ (NFSR.Get(12) & LFSR.Get(8)) ^
(LFSR.Get(13) & LFSR.Get(20)) ^ (NFSR.Get(95) & LFSR.Get(42)) ^ (LFSR.Get(60) &
LFSR.Get(79)) ^ (NFSR.Get(12) & NFSR.Get(95) & LFSR.Get(95));
           }
     }
}



//файл JIBitArray.cs
using System;
using System.Collections.Generic;
using System.Collections;

namespace ApproximationsSearchFramework.Oracles.Additional
{
     [Serializable]
     public class JIBitArray
     {
           //private static ArrayList x = new ArrayList();

           //private ArrayList _Bits = ArrayList.Synchronized(x);

           private ArrayList _Bits = new ArrayList();


           #region Constructors
           public JIBitArray()
           { }
           /*
           public bool this[int index]
           {
               get
               {
                   return this[index];
               }

               set
               {
                   this[index] = value;
               }
           }
           */
           public JIBitArray(byte[] bits)
           {
                string st;
                foreach (byte b in bits)
                {
                     st = FixLength(Convert.ToString(b, 2), 8);
                     AddBits(st);
```

```
        }
    }

    public JIBitArray(string bits)
    {
        AddBits(bits);
    }


    public JIBitArray(int[] bits)
    {
        string st;
        foreach (int i in bits)
        {
            st = FixLength(Convert.ToString(i, 2), 32);
            AddBits(st);
        }
    }

    public JIBitArray(long[] bits)
    {
        string st;
        foreach (long i in bits)
        {
            st = FixLength(Convert.ToString(i, 2), 64);
            AddBits(st);
        }
    }

    public JIBitArray(short[] bits)
    {
        string st;
        foreach (short i in bits)
        {
            st = FixLength(Convert.ToString(i, 2), 16);
            AddBits(st);
        }
    }

    public JIBitArray(bool[] bits)
    {
        foreach (bool b in bits)
        {
            _Bits.Add(b);
        }
    }

    public JIBitArray(int length)
    {
        AddBlock(length, false);
    }

    public JIBitArray(int length, bool defaultValue)
    {
        AddBlock(length, defaultValue);
    }

    private void AddBlock(int length, bool Value)
    {
        for (int i = 0; i < length; i++)
            _Bits.Add(Value);
    }
```

```
#endregion

private string FixLength(string num, int length)
{
     while (num.Length < length)
          num = num.Insert(0, "0");
     return num;
}

private void AddBits(string bits)
{
     foreach (char ch in bits)
     {
          if (ch == '0')
               _Bits.Add(false);
          else if (ch == '1')
               _Bits.Add(true);
          else
               throw (new ArgumentException("bits Contain none 0 1 character"));
     }
}

/// <summary>
/// Convert current System.Collections.JIBitArray to binary string
/// </summary>
/// <returns></returns>
public override string ToString()
{
     string rt = string.Empty;
     foreach (bool b in _Bits)
     {
          if (b == false)
               rt += '0';
          else
               rt += '1';
     }
     return rt;
}

/// <summary>
/// Insert a bit at the spesific position in System.Collections.JIBitArray
/// </summary>
/// <param name="index">The zero-based index of the bit to insert</param>
/// <param name="Value">The Boolean value to assign to the bit</param>
public void Insert(int index, bool Value)
{
     _Bits.Insert(index, Value);
}

/// <summary>
/// Add a bit at the final position in System.Collections.JIBitArray
/// </summary>
/// <param name="Value">The Boolean value to assign to the bit</param>
public void Add(bool Value)
{
     _Bits.Add(Value);
}

/// <summary>
/// Set the bit at the spesific position in System.Collection.JIBitArray
/// </summary>
/// <param name="index">The zero-based index of the bit to set</param>
```

```
        /// <param name="Value">The Boolean value to assign to the bit</param>
        public void Set(int index, bool Value)
        {
            _Bits[index] = Value;
        }

        /// <summary>
        /// Set the bit at the spesific position in System.Collection.JIBitArray
        /// </summary>
        /// <param name="index">The zero-based index of the bit to set</param>
        /// <param name="Value">The Boolean value to assign to the bit</param>
        public JIBitArray Set_ext(int index, bool Value)
        {
            JIBitArray res = SubJIBitArray(0, _Bits.Count);
            res.Set(index, Value);
            return res;
        }

        /// <summary>
        /// Get the value of a bit at the specific position in the
System.Collections.JIBitArray
        /// </summary>
        /// <param name="index">The zero-based index of the value to get</param>
        /// <returns></returns>
        public bool Get(int index)
        {
            return (bool)_Bits[index];
        }

        /// <summary>
        /// Get the number of element actually contained in System.Collections.JIBitArray
        /// </summary>
        public int Count
        {
            get
            {
                return _Bits.Count;
            }
        }

        /// <summary>
        /// Set all bits in System.Collections.JIBitArray to the specific value
        /// </summary>
        /// <param name="Value">The Boolean value to assign to all bits</param>
        public void SetAll(bool Value)
        {
            for (int i = 0; i < _Bits.Count; i++)
            {
                _Bits[i] = Value;
            }
        }

        /// <summary>
        /// Inverts all the bits values in the current System.Collections.JIBitArray, so
        /// that elements set to true are changed to false, and elements set to false
        /// are changed to true
        /// </summary>
        /// <returns></returns>
        public JIBitArray Not()
        {
            JIBitArray RArray = new JIBitArray(_Bits.Count);
            for (int i = 0; i < _Bits.Count; i++)
```

```
                {
                    if ((bool)_Bits[i] == true)
                        RArray.Set(i, false);
                    else
                        RArray.Set(i, true);
                }
                return RArray;
            }

            /// <summary>
            /// Retrives a SubJIBitArray from this instance. The SubJIBitArray start at the
            /// specified bit position and has specified length
            /// </summary>
            /// <param name="index">The index of the start of SubJIBitArray</param>
            /// <param name="length">The number of bits in SubJIBitArray</param>
            /// <returns></returns>
            public JIBitArray SubJIBitArray(int index, int length)
            {
                JIBitArray RArray = new JIBitArray(length);
                int c = 0;
                for (int i = index; i < index + length; i++)
                    RArray.Set(c++, (bool)_Bits[i]);
                return RArray;
            }

            /// <summary>
            /// Performs the bitwise OR operation on the elements in the current
System.Collections.JIBitArray
            /// against the corresponding elements in the specified
System.Collections.JIBitArray
            /// </summary>
            /// <param name="Value">The System.Collections.JIBitArray with which to perform
the bitwise OR operation</param>
            /// <returns></returns>
            public JIBitArray Or(JIBitArray Value)
            {
                JIBitArray RArray;
                int Max = _Bits.Count > Value._Bits.Count ? _Bits.Count : Value._Bits.Count;

                RArray = new JIBitArray(Max);
                RArray._Bits = this._Bits;

                if (Max == RArray._Bits.Count)
                    FixLength(Value, Max);
                else
                    FixLength(RArray, Max);

                for (int i = 0; i < Max; i++)
                    RArray.Set(i, (bool)Value._Bits[i] | (bool)RArray._Bits[i]);

                return RArray;
            }

            /// <summary>
            /// Perform the bitwise AND operation on the elements in the current
System.Collections.JIBitArray
            /// against the corresponding elements in the specified
System.Collections.JIBitArray
            /// </summary>
            /// <param name="Value">The System.Collections.JIBitArray with which to perform
the bitwise OR operation</param>
            /// <returns></returns>
```

```
        public JIBitArray And(JIBitArray Value)
        {
            JIBitArray RArray;
            int Max = _Bits.Count > Value._Bits.Count ? _Bits.Count : Value._Bits.Count;

            RArray = new JIBitArray(Max);
            RArray._Bits = this._Bits;

            if (Max == RArray._Bits.Count)
                FixLength(Value, Max);
            else
                FixLength(RArray, Max);

            for (int i = 0; i < Max; i++)
                RArray.Set(i, (bool)Value._Bits[i] & (bool)RArray._Bits[i]);

            return RArray;
        }

        /// <summary>
        /// Perform the bitwise eXclusive OR operation on the elements in the current
System.Collections.JIBitArray
        /// against the corresponding elements in the specified
System.Collections.JIBitArray
        /// </summary>
        /// <param name="Value">The System.Collections.JIBitArray with which to perform
the bitwise eXclusive OR operation</param>
        /// <returns></returns>
        public JIBitArray Xor(JIBitArray Value)
        {
            JIBitArray _Value = Value.SubJIBitArray(0, Value.Count);

            JIBitArray RArray;
            int Max = _Bits.Count > _Value._Bits.Count ? _Bits.Count :
_Value._Bits.Count;

            RArray = new JIBitArray(Max);
            //RArray._Bits = this._Bits;
            RArray = this.SubJIBitArray(0, this.Count);

            if (Max == RArray._Bits.Count)
                FixLength(_Value, Max);
            else
                FixLength(RArray, Max);

            for (int i = 0; i < Max; i++)
                RArray.Set(i, (bool)_Value._Bits[i] ^ (bool)RArray._Bits[i]);

            return RArray;
        }


        public void FastXor(JIBitArray Value)
        {
            for (int i = 0; i < this._Bits.Count; i++)
            {
                if (this.Get(0) == Value.Get(i))
                {
                    this.Set(i, false);
                }
                else
                {
```

```
                this.Set(i, true);
            }

        }
    }


    #region Array Convertors
    /// <summary>
    /// Convert current System.Collections.JIBitArray to a long array
    /// </summary>
    /// <returns></returns>
    public long[] GetLong()
    {
        int ArrayBound = (int)Math.Ceiling((double)this._Bits.Count / 64);
        long[] Bits = new long[ArrayBound];
        JIBitArray Temp = new JIBitArray();
        Temp._Bits = this._Bits;
        Temp = FixLength(Temp, ArrayBound * 64);
        for (int i = 0; i < Temp._Bits.Count; i += 64)
        {
            Bits[i / 64] = Convert.ToInt64(Temp.SubJIBitArray(i, 64).ToString(), 2);
        }
        return Bits;
    }

    /// <summary>
    /// Convert current System.Collections.JIBitArray to a int array
    /// </summary>
    /// <returns></returns>
    public int[] GetInt()
    {
        int ArrayBound = (int)Math.Ceiling((double)this._Bits.Count / 32);
        int[] Bits = new int[ArrayBound];
        JIBitArray Temp = new JIBitArray();
        Temp._Bits = this._Bits;
        Temp = FixLength(Temp, ArrayBound * 32);

        for (int i = 0; i < Temp._Bits.Count; i += 32)
        {
            Bits[i / 32] = Convert.ToInt32(Temp.SubJIBitArray(i, 32).ToString(), 2);
        }
        return Bits;
    }

    public int[] GetIntBits()
    {
        int[] result = new int[this.Count];

        for (int i = 0; i < this.Count; i++)
        {
            if (this.Get(i))
            {
                result[i] = 1;
            }
        }

        return result;
    }


    /// <summary>
```

```
/// Convert current System.Collections.JIBitArray to a short array
/// </summary>
/// <returns></returns>
public short[] GetShorts()
{
     int ArrayBound = (int)Math.Ceiling((double)this._Bits.Count / 16);
     short[] Bits = new short[ArrayBound];
     JIBitArray Temp = new JIBitArray();
     Temp._Bits = this._Bits;
     Temp = FixLength(Temp, ArrayBound * 16);

     for (int i = 0; i < Temp._Bits.Count; i += 16)
     {
          Bits[i / 16] = Convert.ToInt16(Temp.SubJIBitArray(i, 16).ToString(), 2);
     }
     return Bits;
}

/// <summary>
/// Convert current System.Collections.JIBitArray to a byte array
/// </summary>
/// <returns></returns>
public byte[] GetBytes()
{
     int ArrayBound = (int)Math.Ceiling((double)this._Bits.Count / 8);
     byte[] Bits = new byte[ArrayBound];
     JIBitArray Temp = new JIBitArray();
     Temp._Bits = this._Bits;
     Temp = FixLength(Temp, ArrayBound * 8);

     for (int i = 0; i < Temp._Bits.Count; i += 8)
     {
          Bits[i / 8] = Convert.ToByte(Temp.SubJIBitArray(i, 8).ToString(), 2);
     }
     return Bits;
}
#endregion

/// <summary>
/// Shift the bits of current System.Collections.JIBitArray as specified number to
/// left
/// </summary>
/// <param name="count">Specific number to shift left</param>
/// <returns></returns>
public JIBitArray ShiftLeft(int count)
{
     JIBitArray RArray = new JIBitArray();
     RArray._Bits = this._Bits;

     for (int i = 0; i < count; i++)
     {
          RArray._Bits.RemoveAt(0);
          RArray._Bits.Add(false);

     }
     return RArray;
}

public void ShiftLeft()
{
     this._Bits.RemoveAt(0);
     this._Bits.Add(false);
```

```
        }


        /// <summary>
        /// Shift the bits of current System.Collections.JIBitArray as specified number to
        /// right
        /// </summary>
        /// <param name="count">Specific number to shift right</param>
        /// <returns></returns>
        public JIBitArray ShiftRight(int count)
        {
             JIBitArray RArray = new JIBitArray();
             RArray._Bits = this._Bits;
             for (int i = 0; i < count; i++)
             {
                  RArray._Bits.RemoveAt(RArray._Bits.Count - 1);
                  RArray._Bits.Insert(0, false);
             }
             return RArray;
        }

        /// <summary>
        /// Remove zero's of begining of current System.Collections.JIBitArray
        /// </summary>
        /// <returns></returns>
        public JIBitArray RemoveBeginingZeros()
        {
             JIBitArray RArray = new JIBitArray();
             RArray._Bits = this._Bits;
             while (RArray._Bits.Count != 0 && (bool)RArray._Bits[0] == false)
                  RArray._Bits.RemoveAt(0);
             return RArray;
        }


        public JIBitArray FlipBit(int index)
        {
             JIBitArray RArray = new JIBitArray();
             RArray = this.SubJIBitArray(0, this.Count);
             RArray.Set(0, !this.Get(index));
             return RArray;
        }


        public JIBitArray Concat(JIBitArray bits)
        {
             JIBitArray RArray = new JIBitArray();
             RArray = this.SubJIBitArray(0, this.Count);

             for (int i = 0; i < bits.Count; i++)
             {
                  RArray.Add(bits.Get(i));
             }
             return RArray;
        }


        #region Static
        /// <summary>
        /// Insert enough zero at the begining of the specified
System.Collections.JIBitArray to
```

```
          /// make it's lenght to specified length
          /// </summary>
          /// <param name="Value">The System.Collections.JIBitArray with wich to insert zero
to begining</param>
          /// <param name="length">The number of bits of Value after inserting</param>
          /// <returns></returns>
          public static JIBitArray FixLength(JIBitArray Value, int length)
          {
               if (length < Value._Bits.Count)
                    throw (new ArgumentException("length must be equal or greater than
Bits.Length"));
               while (Value._Bits.Count < length)
                    Value._Bits.Insert(0, false);
               return Value;
          }
          #endregion

          #region Operators
          public static JIBitArray operator &(JIBitArray Bits1, JIBitArray Bits2)
          {
               return Bits1.And(Bits2);
          }

          public static JIBitArray operator |(JIBitArray Bits1, JIBitArray Bits2)
          {
               return Bits1.Or(Bits2);
          }

          public static JIBitArray operator ^(JIBitArray Bits1, JIBitArray Bits2)
          {
               return Bits1.Xor(Bits2);
          }

          public static JIBitArray operator >>(JIBitArray Bits, int count)
          {
               return Bits.ShiftRight(count);
          }

          public static JIBitArray operator <<(JIBitArray Bits, int count)
          {
               return Bits.ShiftLeft(count);
          }
          #endregion

          public override bool Equals(object obj)
          {
               JIBitArray compare = (JIBitArray)obj;
               if (compare.Count != this.Count)
               {
                    return false;
               }

               if (this.Xor(compare).RemoveBeginingZeros().Count != 0)
               {
                    return false;
               }
               return true;
          }

          public override int GetHashCode()
          {
               return this._Bits.GetHashCode() * 43;
```

```
            }

        }
    }
    //файл Grain_FisherOracle.cs
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    using ApproximationsSearchFramework.Oracles.Additional;
    using ApproximationsSearchFramework.Oracles;

    namespace ApproximationsSearchFramework
    {
        class Grain_FisherOracle : IOracle
        {
            //key bit-length
            public const int KEY_BIT_LENGTH = 128;

            //iv bit-length
            public const int IV_BIT_LENGTH = 96;

            public int l0 { get { return KEY_BIT_LENGTH; } }

            public int l1 { get { return IV_BIT_LENGTH; } }



            public static readonly int[] weakIVbits = { 2, 6, 8, 55, 58, 78, 90 };
            public const int ROUNDS = 180;

            private Grain128 cipher;

            private JIBitArray keyBits;
            private JIBitArray ivBits;

            private byte[] key;
            private byte[] iv;


            public Grain_FisherOracle()
            {
                keyBits = new JIBitArray(l0, false);
                ivBits = new JIBitArray(l1, false);

                key = new byte[l0 / 8];
                iv = new byte[l1 / 8];

                cipher = new Grain128();
            }


            public void Init()
            {

            }

            public bool GetKeystreamBit(byte[] x)
            {
```

```
            Array.Copy(x, 0, key, 0, l0 / 8);
            Array.Copy(x, l0 / 8, iv, 0, l1 / 8);

            keyBits = new JIBitArray(key);
            ivBits = new JIBitArray(iv);


            JIBitArray[] L = BuildSubspace(ivBits, weakIVbits);

            int count = 0;
            bool outbit = false;

            for (int i = 0; i < L.Length; i++)
            {
                cipher.Init(keyBits, ivBits.Xor(L[i]), ROUNDS);
                outbit = cipher.Generate_Gamma_Bits(1).Get(0);

                if (outbit)
                {
                    count++;
                }
            }

            return count % 2 == 1 ? true : false;
        }



        //Creating L (subspace based on weak bits in IV)
        private JIBitArray[] BuildSubspace(JIBitArray iv_bits, int[] weak_bits)
        {
            JIBitArray[] L_subspace = new JIBitArray[(int)Math.Pow((double)2,
(double)weak_bits.Length)];
            JIBitArray[] vectors = VectorTable(weak_bits.Length);
            for (int i = 0; i < (int)Math.Pow((double)2, (double)weak_bits.Length); i++)
            {
                L_subspace[i] = new JIBitArray(iv_bits.Count);
                for (int j = 0; j < weak_bits.Length; j++)
                {
                    L_subspace[i].Set(weak_bits[j], vectors[i].Get(j));
                }
            }
            return L_subspace;
        }

        //build table of all vectors of k variables
        private JIBitArray[] VectorTable(int k)
        {
            int rows = (int)Math.Pow((double)2, (double)k);
            bool[] tmp = new bool[k];
            JIBitArray[] result = new JIBitArray[rows];
            string x = string.Empty;
            char[] characters = new char[0];
            bool[] vector = new bool[0];

            for (int i = 0; i < rows; i++)
            {
                x = Convert.ToString(i, 2);
                x = x.PadLeft(k, '0');
                characters = x.ToCharArray();
                vector = new bool[characters.Length];
```

```
                    for (int j = 0; j < characters.Length; j++)
                    {
                        if (characters[j] == '1')
                        {
                            vector[j] = true;
                        }
                        else
                        {
                            vector[j] = false;
                        }
                    }
                    result[i] = new JIBitArray(vector);
                }
                return result;
            }

            public override string ToString()
            {
                return "Grain-128 Fisher oracle";
            }
        }
    }


    //файл IOracle.cs
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    namespace ApproximationsSearchFramework
    {
        public interface IOracle
        {
            //Bit-length of key
            int l0 { get; }

            //Bit-length of iv
            int l1 { get; }

            void Init();
            bool GetKeystreamBit(byte[] x);

        }
    }


    //файл RC4_Oracle.cs
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;

    namespace ApproximationsSearchFramework
    {
        class RC4_Oracle : IOracle
        {
            public int l0
            {
                get { throw new NotImplementedException(); }
            }
```

```
            public int l1
            {
                get { throw new NotImplementedException(); }
            }

            public void Init()
            {
                throw new NotImplementedException();
            }

            public bool GetKeystreamBit(byte[] x)
            {
                throw new NotImplementedException();
            }
        }
}



//файл Snow20_Oracle.cs
using System;

using System.Runtime.InteropServices;

namespace ApproximationsSearchFramework
{
    class Snow20_Oracle : IOracle
    {
        //key bit-length
        public const int KEY_BIT_LENGTH = 128;

        //iv bit-length
        public const int IV_BIT_LENGTH = 128;

        public int l0 { get { return KEY_BIT_LENGTH; } }

        public int l1 { get { return IV_BIT_LENGTH; } }

        public const string DLL_NAME = @"D:\Docs\diser_programms\SNOW\DLL\native-
snow2.0.dll";

        [DllImport(DLL_NAME, EntryPoint = "ECRYPT_keysetup")]
        public static extern void KeySetup(IntPtr ctx,
                                           IntPtr key,
                                           int keyBitLength,
                                           int ivBitLength);


        [DllImport(DLL_NAME, EntryPoint = "ECRYPT_ivsetup")]
        public static extern void IVsetup(IntPtr ctx, IntPtr iv);



        [DllImport(DLL_NAME, EntryPoint = "ECRYPT_process_bytes")]
        public static extern void ProcessBytes(int action, IntPtr ctx, IntPtr input,
IntPtr output, int messageByteLength);



        [StructLayout(LayoutKind.Sequential)]
        public struct ECRYPT_CTX
        {
```

```
        int keysize;

        [MarshalAs(UnmanagedType.ByValArray, SizeConst = 32)]
        byte[] key;

        int s15, s14, s13, s12, s11, s10, s9, s8, s7, s6, s5, s4, s3, s2, s1, s0;
        int r1, r2;

    }

    byte[] key;
    byte[] iv;
    byte[] input;

    ECRYPT_CTX snow2_0;


    IntPtr keyPtr;
    IntPtr ivPtr;
    IntPtr inputPtr;
    IntPtr outputPtr;

    IntPtr snow2_0Ptr;




    public Snow20_Oracle()
    {
        keyPtr = new IntPtr();
        ivPtr = new IntPtr();
        inputPtr = new IntPtr();
        outputPtr = new IntPtr();
        snow2_0Ptr = new IntPtr();

        key = null;
        iv = null;
        input = new byte[1];

        snow2_0 = new ECRYPT_CTX();
    }


    public void Init()
    {


        key = new byte[KEY_BIT_LENGTH / 8];
        iv = new byte[IV_BIT_LENGTH / 8];

        keyPtr = Marshal.AllocHGlobal(KEY_BIT_LENGTH / 8);
        ivPtr = Marshal.AllocHGlobal(IV_BIT_LENGTH / 8);
        inputPtr = Marshal.AllocHGlobal(1);
        outputPtr = Marshal.AllocHGlobal(1);

        snow2_0Ptr = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(ECRYPT_CTX)));
        Marshal.StructureToPtr(snow2_0, snow2_0Ptr, true);

    }
```

```
            public bool GetKeystreamBit(byte[] inputVector)
            {
                Array.Copy(inputVector, 0, key, 0, key.Length);
                Array.Copy(inputVector, key.Length, iv, 0, iv.Length);

                Marshal.Copy(key, 0, keyPtr, key.Length);

                Marshal.Copy(iv, 0, ivPtr, iv.Length);

                Marshal.Copy(input, 0, inputPtr, input.Length);



                //BITS LENGTH!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                KeySetup(snow2_0Ptr, keyPtr, key.Length * 8, iv.Length * 8);
                IVsetup(snow2_0Ptr, ivPtr);

                //BYTES LENGTH!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                ProcessBytes(0, snow2_0Ptr, inputPtr, outputPtr, 1);


                return GetBit(Marshal.ReadByte(outputPtr));
            }






            private bool GetBit(byte keystream)
            {
                //most significant bit
                //return (keystream & 0x80) != 0;

                //least significant bit
                return (keystream & 0x01) != 0;
            }

            public override string ToString()
            {
                return "SNOW-2.0";
            }

        }
    }

    //файл ApproximationSearchMethod.cs
    using System;
    using System.Security.Cryptography;

    using ApproximationsSearchFramework.Algorithms;

    namespace ApproximationsSearchFramework
    {
        /// <summary>
        /// See theoretical description of the new proposed method in the article:
        ///
        /// "A METHOD OF FINDING THE ALGEBRAIC DEGENERATE APPROXIMATIONS OF BOOLEAN FUNCTIONS
        /// FOR A CONSTRUCTION OF STATISTICAL ATTACKS ON SINCHRONOUS STREAM CIPHERS"
        ///
        /// by Anton Alekseychuk, Sergey Konushok, Artem Storozhuk.
        /// </summary>
```

```
     public class ApproximationsSearchMethod
     {
          public static RNGCryptoServiceProvider randomSource = new
RNGCryptoServiceProvider();

          private IOracle oracle;


          public ApproximationsSearchMethod(IOracle oracle)
          {
               if (oracle == null)
               {
                    throw new NullReferenceException("Oracle is not specified");
               }

               this.oracle = oracle;
          }

          internal void PerformStepOne()
          {
               AlgorithmOne alg1 = new AlgorithmOne(oracle);
               alg1.Perform();
          }

          internal void PerformStepTwo()
          {
               AlgorithmTwo alg2 = new AlgorithmTwo(oracle);
               alg2.Perform();
          }

          internal void PerformStepThree()
          {
               throw new NotImplementedException();
          }

          internal void PerformStepFour()
          {
               throw new NotImplementedException();
          }

          internal void PerformStepFive()
          {
               throw new NotImplementedException();
          }
     }
}


//файл Start.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Security.Cryptography;

namespace ApproximationsSearchFramework
{
     class Start
     {
          static void Main(string[] args)
          {
               IOracle oracle = null;
```

```
            Selecting_oracle:

                  Console.WriteLine("Select an ORACLE function to test");
                  Console.WriteLine("1 - Grain 180 rounds (made by Fisher)");
                  Console.WriteLine("2 - RC4");
                  Console.WriteLine("3 - SNOW 2.0");

                  string userSelect = Console.ReadLine();
                  Console.WriteLine();

                  switch (userSelect)
                  {
                        case "1": oracle = new Grain_FisherOracle(); break;
                        case "2": oracle = new RC4_Oracle(); break;
                        case "3": oracle = new Snow20_Oracle(); break;

                        case "q": return;

                        default :
                              //Console.WriteLine("Please select only 1,2 or 3");
                              goto Selecting_oracle;
                  }




                  ApproximationsSearchMethod method = new ApproximationsSearchMethod(oracle);

                  #region STEP 1 - selecting threshold for vectors rejection (MU^2 upper bound
evaluation)

                  //method.PerformStepOne();

                  #endregion


                  #region STEP 2 - searching low-weight vectors for specified oracle

                  method.PerformStepTwo();

                  #endregion


                  #region STEP 3 - building M matrix

                  //method.PerformStepThree();

                  #endregion


                  #region STEP 4 - evaluating distance to subspace generated by found vectors

                  //method.PerformStepFour();

                  #endregion


                  #region STEP 5 - building approximation
```

```
            //method.PerformStepFive();

            #endregion
        }



        internal static void Time()
        {
            Console.Write(DateTime.Now.Hour + ":");
            Console.Write(DateTime.Now.Minute + ":");
            Console.Write(DateTime.Now.Second + ":");
            Console.WriteLine(DateTime.Now.Millisecond);
        }
    }
}
```

“ЗАТВЕРДЖУЮ”
Виконавчий директор АТ «ІТ»
_____ В.Д. Кравченко
«25» 01 2016 р.

АКТ
впровадження результатів дисертаційних досліджень

Сторожука Артема Юрійовича
у Приватному акціонерному товаристві «Інститут інформаційних технологій»

Комісія у складі голови комісії, головного конструктора, доктора технічних наук, професора Горбенка І.Д. і членів комісії, заступника головного конструктора з БС, кандидата технічних наук, професора Качко О.Г., і начальника відділу АЗЗІ, кандидата технічних наук Бобуха В.А. встановила, що у Приватному акціонерному товаристві «Інститут інформаційних технологій» впроваджені наступні результати, які одержані Сторожуком Артемом Юрійовичем у процесі виконання дисертаційних досліджень.

1. Метод пошуку алгебраїчно вироджених наближень булевих функцій для побудови статистичних атак на синхронні потокові шифри.
2. Метод обґрунтування стійкості синхронних потокових шифрів відносно узагальненої статистичної атаки.
3. Метод оцінювання відносної відстані між зрівноваженою булевою функцією та множиною k-вимірних функцій.

Ефект від впровадження зазначених наукових результатів полягає в тому, що вони дозволяють підвищити (у певних випадках – в 1000 та більше разів) ефективність відомих алгоритмів побудови алгебраїчно вироджених наближень булевих функцій, оцінювати та обґрунтовувати практичну стійкість синхронних потокових шифрів чи їх компонентів відносно узагальненої статистичної атаки і, таким чином, підвищити якість експертних рішень щодо практичного використання перспективних алгоритмів потокового шифрування в Україні.

Голова комісії, д.т.н., проф. І.Д. Горбенко

Члени комісії:
к.т.н., проф. О.Г. Качко

к.т.н. В.А. Бобух

"ЗАТВЕРДЖУЮ"
Заступник директора департаменту-
начальник управління
Служби зовнішньої розвідки України

Романович К.О.

"30" 09 2016 року

АКТ

впровадження результатів досліджень дисертаційної роботи
Сторожука Артема Юрійовича в науково-дослідній роботі "Дослідження та застосування сучасних математичних методів аналізу окремих перетворень у системах криптографічного захисту інформації" (шифр "Мокрель")

Комісія у складі голови комісії Сединкіна С.К. та членів комісії: Черевко Ю.П., Гудзенко С.В. з'ясувала, що в Службі зовнішньої розвідки України в результаті виконання науково-дослідної роботи "Дослідження та застосування сучасних математичних методів аналізу окремих перетворень у системах криптографічного захисту інформації " (шифр "Мокрель") вперше впроваджено отриманий Сторожуком Артемом Юрійовичем такий науковий результат: **метод пошуку алгебраїчно вироджених наближень булевих функцій для побудови статистичних атак на синхронні потокові шифри.**

Ефект від впровадження зазначеного наукового результату полягає в тому, що він дозволяє:

– суттєво розширити клас булевих функцій, які можуть бути використані для побудови статистичних атак на синхронні потові шифри;

– розробити та програмно реалізувати поліноміальні ймовірнісні алгоритми (перевірки існування, пошуку чи побудови наближень), які можуть бути застосовані на практиці до функцій-оракулів від декількох десятків чи сотень змінних;

– переконуватися у відсутності шуканих наближень і, таким чином, скорочувати час, що витрачається на пошук вразливостей сучасних потокових шифрів відносно низки відомих статистичних атак.

Голова комісії: _____ Сединкін С.К.

Члени комісії:
к.т.н. _____ Черевко Ю.П.

к.ф.-м.н. _____ Гудзенко С.В.

"29" 09 2016 року

"ЗАТВЕРДЖУЮ"
Заступник директора департаменту-
начальник управління
Служби зовнішньої розвідки України

Романович К.О.

"30" 09 2016 року

АКТ
впровадження результатів досліджень дисертаційної роботи
Сторожука Артема Юрійовича в науково-дослідній роботі "Дослідження сучасних алгебраїчно-ймовірнісних методів криптоаналізу симетричних та асиметричних криптосистем і застосування цих методів до окремих систем криптографічного захисту інформації" (шифр "Севрюга")

Комісія у складі голови комісії Сединкіна С.К. та членів комісії: Черевко Ю.П., Гудзенко С.В. з'ясувала, що в Службі зовнішньої розвідки України в результаті виконання науково-дослідної роботи "Дослідження сучасних алгебраїчно-ймовірнісних методів криптоаналізу симетричних та асиметричних криптосистем і застосування цих методів до окремих систем криптографічного захисту інформації" (шифр "Севрюга") вперше впроваджено отримані Сторожуком Артемом Юрійовичем такі наукові результати:

1. Статистичні атаки на синхронні потокові шифри на основі алгебраїчно вироджених наближень булевих функцій.
2. Метод побудови списку k-вимірних наближень булевих функцій, що використовуються у синхронних потокових шифрах.

Ефект від впровадження зазначених наукових результатів полягає в тому, що вони дозволяють:

– запропонувати статистичні атаки на синхронні потокові шифри, які узагальнюють низку раніше відомих, зокрема, так звану атаку FKM і кубічну атаку;

– охарактеризувати клас ключів, які можуть бути відновлені за допомогою узагальненої статистичної атаки, та отримати явний вираз її трудомісткості;

– розробити ймовірнісний алгоритм побудови булевих функцій від меншої кількості змінних, що використовуються для визначення наближень при проведенні узагальненої статистичної атаки;

– підвищити (у певних випадках – в 1000 та більше разів) ефективність відомих алгоритмів побудови алгебраїчно вироджених наближень булевих функцій;

– запропонувати статистичну атаку на редуковану версію шифру Grain-128, яка є в 2^{27} разів ефективніше у порівнянні з аналогічною раніше відомою атакою.

Голова комісії: _____ Сединкін С.К.

Члени комісії:
к.т.н. _____ Черевко Ю.П.

к.ф.-м.н. _____ Гудзенко С.В.

"29" 09 2016 року