НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ ЗАХИСТУ ІНФОРМАЦІЇ

«На правах рукопису»
УДК 519.2+519.688

«До захисту допущено»
В.о. завідувача кафедрою
_____ М.М.Савчук
(підпис) (ініціали, прізвище)
“15” травня 2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 113 «Прикладна математика»

на тему: Уточнений метод оцінювання імовірностей диференціалів немарковських AES – подібних шифрів

Виконав (-ла): студент (-ка) 2 курсу, групи _____ФІ-63М_____
(шифр групи)

Байбуз Микола Андрійович

Керівник к.т.н. Яковлєв С.В.

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент к.т.н. Ковтун В.Ю.

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018року

РЕФЕРАТ

Дипломну роботу виконано на 43 аркушах, вона містить 2 додатки та перелік посилань на використані джерела з 16 найменувань. У роботі наведено 3 рисунки та 4 таблиці.

Метою даної дипломної роботи є аналіз, уточнення та застосування методів дослідження марковських SP-мереж на стійкість до диференціального криптоаналізу.

Обєктом дослідження є інформаційні процеси в системах криптографічного захисту.

Предметом дослідження є алгоритми оцінювання SP-мереж на стійкість до диференціального криптоаналізу.

В роботі проводиться уточнення та застосування методів для оцінки стійкості SP-мереж до диференціального криптоаналізу на прикладі шифру ДСТУ 7624:2014.

Основні положення дипломної роботи опубліковано у вигляді тез доповіді на Міжнародній науково-практичній конференції БІВІТС 2016 та Всеукраїнській науково-практичній коференції ТІППФМТІ 2016.

Ключові слова: SP-мережа, диференціальний криптоаналіз, шифр ДСТУ 7624:2014, диференціальна ймовірність.

РЕФЕРАТ

Дипломную работу выполнено на 43 листах, она содержит 2 приложения и перечень использованных источников из 16 наименований. В работе приведены 3 рисунки и 4 таблицы.

Целью данной работы является анализ, уточнение и применение методов исследования марковских SP-сетей на устойчивость к дифференциальному криптоанализу.

Объектом исследования являются процессы в системах криптографической защиты.

Предметом исследования являются алгоритмы оценки SP-сетей на устойчивость к дифференциальному криптоанализу.

В работе проводится уточнение и применение методов для оценки устойчивости SP-сетей к дифференциальному криптоанализу на примере шифра ДСТУ 7624:2014.

Основные положения дипломной работы опубликованы в виде тезисов на Международной научно-практической конференции БИВИТС 2016 и Всеукраинской научно-практической коференции ТИППФМТИ 2016.

Ключевые слова: SP-сеть, дифференциальный криптоанализ, шифр ГОСТ 7624:2014, дифференциальная вероятность.

ABSTRACT

The thesis is presented in 43 pages. It contains 2 appendixes and bibliography of 16 references. Four figures and 2 tables are given in the thesis.

The goal of the thesis is the analysis, specification and application of research methods Markov SP-networks for resistance to differential cryptanalysis.

The object of research is the information processes in cryptographic protection systems.

The subject of research is estimation of SP-networks algorithms for resistance to differential cryptanalysis.

In the presented thesis the Markov SP-networks resistance to differential cryptoanalysis is assessed, taking the DSTU 7624:2014 cipher as the illustrative example.

Main ideas of the thesis were published in the Proceedings of the International Practical and Technical Conference БІВІТС 2016.

Keywords: SP-network, differential cryptanalysis, cipher DSTU 7624:2014, differential probability.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

V_u — простір u-бітних векторів.

$(V_u)^m$ — простір m бітних векторів з u-бітними координатами.

$wt(x)$ — кількість не нульових елементів вектору x.

$[P]$ — дужки Айверсона: $[P]$ дорівнює 1, якщо P – істинне, та 0, якщо P – хибне.

$\overline{\sum_x}$ — середня сума: якщо $x \in X$, то $\overline{\sum_x} \equiv \frac{1}{|x|}$.

ВСТУП

Одним із потужних методів сучасного криптоаналізу симетричних шифрів є диференціальний криптоаналіз, запропонований в 1991 році Біхамом і Шаміром [8]. Стійкість до нього є одним з обовязкових критеріїв при розробці та аналізі сучасних блочних шифрів. Задача доказової стійкості схем и шифрування до диференціального аналізу була сформульована К. Ніберг та Л.Р. Кнудсеном [16]. В сучасній теорії розрізняють теоретичну та практичну стійкість до диференціального аналізу. Для дослідження практичної стійкості необхідно знайти диференціальну характеристику, що має найбільшу ймовірність, для дослідження теоретичної стійкості – диференціал, що має найбільшу ймовірність. Хоча проведення атаки за допомогою високоймовірних диференціальних характеристик є найбільш легким і зручним способом криптоаналізу, але забезпечення практичної стійкості не гарантує стійкості в цілому.

Відповідна задача наразі не розвязана в загальному виді, існуючі моделі та методи застосовні лише в окремих випадках та часто повертають неадекватні з точки зору практики результати. Отже, описана проблематика вимагає глибоких та ретельних досліджень.

Обєктом дослідження даної роботи є інформаційні процеси в системах криптографічного захисту.

Предметом дослідження даної роботи є алгоритми автоматичного оцінівання стійкості марковських SP-мереж до диференціально криптоаналізу.

Метою даної роботи є розробка, аналіз, уточнення та застосування нових методів дослідження марковських SP-мереж на стійкість до диференціального криптоаналізу.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

- виконати аналіз структури марковської SP-мережі;
- дослідити методи побудови матриць верхніх меж імовірностей диференеціалів;
- уточнити методи побудови матриць верхніх меж імовірностей диференеціалів;

- реалізувати алгоритм знаходження верхніх меж імовірностей диференеціалів та експериментально знайти такі імовірності для шифру ДСТУ 7624:2014 та його модифікацій.

Методи дослідження грунтуються на теоретичному та практичному аналізі структури марковської SP-мережі.

Наукова новизна полягає в тому, що уточнені методи оцінки стійкості марковських SP-мереж на стійкість до диференціального криптоаналізу.

Практична значимість даної роботи полягає в тому, що запропоновані методи дозволяють досліджувати існуючі та створювати нові надійні алгоритми шифрування на основі марковських SP-мереж.

1 ФОРМАЛЬНИЙ ОПИС SP-МЕРЕЖ

В цьому розділі наведено теоретичні відомості та основні теоречні поняття з диференціального криптоаналізу. Наведено формальний опис SP-мереж та теоретичні результати щодо стійкості SP-мереж. Розглянуто алгоритм шифрування ДСТУ 7624:2014 та наведені наявні оцінки стійкості цього алгоритму.

1.1 Сутність дифренціального криптоаналізу

Диференціальний криптоаналіз відноситься до атак останнього раунду. Оновною метою проведення аналізу є встановлення раундового ключа останнього раунду k_r. Розглянемо схему атаки на ітеративний блочний шифр побудовану за допомогою диференціального криптоаналізу.

Нехай на просторі $(V_u)^m$ задано деякі операції $\circ$ та $\bullet$ блочного вигляду які задають структуру абелевої групи. Нехай $L\colon (V_u)^m \to (V_u)^m$ – лінійне перетворення (відносно операції $\circ$), тоді індексом розгалудження L називається величина B:

$$B = B(L) = \min_{x \neq 0} \{wt(x) + wt(L(x))\}.$$

Нехай V_q простір з операціями $\circ$ та $\bullet$. E – ітеративний блоний шифр. $E = F_{1,r-1} \times f_r$ – композиція перетворень, де f_r – останній раунд, $F_{1,r-1}$ – всі інші раунди. $(X,X^{'})$ – пари відкритих текстів, для яких $X^{'} = X \circ a$ для фіксованого a. $(Y,Y^{'})$ – відповідні парам відкритих текстів, для яких $Y = F_{1,r-1}(X), Y^{'} = F_{1,r-1}(X^{'})$. Для деякого b виконується $Y^{'} = Y \bullet b$ для деякого a з високою імовірністю p $(p \gg 2^{-q})$.

Побудуємо статичний розпізнавач для ключа k_r:

а) Накопичуємо деяку кількість пар випадкових відкритих текстів $(X,X^{'})$

таких, що $X^{'} = X \circ a$ та відповідних їм пар шифртекстів $(C,C^{'})$;

б) Розшифровуємо пари $(C,C^{'})$ на один раунд та одержуємо $(Y,Y^{'})$, для кожного кандидата в ключі k;

в) Перевіряємо імовірність гіпотези $Y^{'} = Y \bullet b$, якщо імовірність близька до p, то ключ вгадано вірно;

Також можна поудувати структурний розаізнавач: із значень $(C,C^{'})$ та припущення, що $Y^{'} = Y \bullet b$, для кожної пари обчислюємо можливі значення ключа k_r. Коли одне із значень почне домінувати атака припиняється.

Для SP-мереж, в яких додавання із ключем зазвичай відбувається за допомогою операції $\oplus$, продуктивним є підхід Біхама та Шаміра [8, 9]. В цій роботі розглядались пари текстів із фіксованою різницею відносно операції $\oplus$. Вивчення диференціального криптоаналізу із використанням інших операцій майже не проводилось.

Біхам та Шамір показали, що для успішного розпізнавання необхідно $O(p^{-1})$ відкритих текстів.

1.1.1 Основні теоретичні поняття диференціального криптоаналізу

Диференціал булевої функції f відносно операції $(\circ,\bullet)$ – це пара двійкових векторів (a,b), для яких існує значення x таке, що виконується співвідношення:

$$f(x \circ a) \bullet (f(x))^{-1} = b,$$

де через $(f(x))^{-1}$ позначено елемент, обернений до $f(x)$ відносно операції $\bullet$.

Імовірність диференціала булевої функції f відносно операції $(\circ,\bullet)$ визначається за такою формулою:

$$d^f(a,b) = \sum_x [f(x \circ a) \bullet (f(x))^{-1} = b]$$

Поняття введені Ковальчук [11] наведені нижще.

Середня за ключами імовірність диференціала булевої функції f_k відносно операції ($\circ$,$\bullet$) в точці x:

$$d^{f_k}(x,a,b) = \sum_k [f_k(x \circ a) \bullet (f_k(x))^{-1} = b]$$

Максимальна диференціальна імовірність параметризованої ключем функції f_k, враховуючи всі точки входу:

$$MDP(f_k) = \max_{a \neq 0,b,x} d^{f_k}(x,a,b).$$

Середня за ключами імовірність диференціалу (a,b) параметризованої ключем функції f_k:

$$EDP^{f_k}(a,b) = \overline{\sum_x} d^{f_k}[k](a,b).$$

Максимальна середня імовірність диференціала (a,b) булевої функції f_k:

$$MEDP(E) = \max_{a \neq 0,b} EDP^{f_k}(a,b).$$

Нехай E – ітеративний блочний шифр, що складається з послідовних раундових перетворень $f_{k_1}^{(1)}, f_{k_2}^{(2)}, f_{k_3}^{(3)}, ..., f_{k_r}^{(r)}$. Раундові ключі k_i вважаються незалежними та рівномірно розподіленими.

Диференціальна характеристика шифру E – послідовність бітових векторів Ω $(\omega_1,\omega_2,\omega_3,...,\omega_r)$, де всі $\omega_i \in V_q \backslash \{0\}$. Диференціальна характеристика розглядається як послідовність змін даних між раундами під час шифрування, тобто якщо подати на вхід два повідомлення X_0 та $X_0^{'}$ такі, що $X_0 \circ (X_0^{'})^{-1} = \omega_0$, то матимемо $X_1 \circ (X_1^{'})^{-1} = \omega_1$, $X_2 \circ (X_2^{'})^{-1} = \omega_2$,..., $X_r \circ (X_r^{'})^{-1} = \omega_r$. З формальної точки зору диференціальною характеристикою шифру може бути довільна послідовність ненульових двійкових векторів потрібної довжини.

Середня за ключами імовірність диференціальної характеристики Ω в точці X_0:

$$DP^E(\Omega,X_0) = \overline{\sum_{k_1}\sum_{k_2}\sum_{k_3}}\ldots\overline{\sum_{k_r}}\prod_{i=1}^{r}[f_{k_i}^{(i)}(X_{i-1}\circ\omega_{i-1})\circ(f_{k_i}^{(i)}(X_{i-1}))^{-1} = \omega_i].$$

Середня імовірність диференціальної характеристики Ω:

$$EDP^E(\Omega) = \overline{\sum_X} DP^E(\Omega,X).$$

В сучасній теорії, вслід за Кандоюта [10] і Кнудсеном [16], розрізняють теоретичну та практичну стійкість до диференціального аналізу. Блочний шифр є теоретично стійким до диференціального аналізу, якщо виконується нерівність:

$$MEDP(E) \leq 2^{-c},$$

для деякого порогового значення c.

Блочний шифр є практично стійким до диференціального аналізу, якщо виконується нерівність:

$$\max_{\Omega} EDP^E(\Omega) \leq 2^{-c},$$

для деякого порогового значення c.

Теоретична стійкість показує, що складність проведення диференціальної атаки із використанням багатораундових диференціалів в середньому складатиме щонайменше 2^c операцій, а практична стійкість – що складність проведення диференціальної атаки із використанням невеликої кількості диференціальних характеристик складатиме щонайменше 2^c операцій.

Практична стійкість шифру гарантує захист від найпоширенішого та (на наш час) самого потужного методу проведення диференціального аналізу, однак не гарантує стійкості в цілому. Втім, оскільки атака із використанням диференціальних характеристик є відносно легкою в проведенні, обидва параметри (як теоретичної, так і практичної стійкості) є важливими. В наш час, з огляду на наявні

обчислювальні потужності, шифри вважаються стійкими при $c \geq 80$.

Аналіз неможливих диференціалів використовує для побудови атаки диференціали із імовірністю 0 (тобто диференціали, які ніколи не виникають під час шифрування). Аналіз неможливих диференціалів показує важливість оцінювання диференціалів шифру не тільки зверху, але й знизу; однак, на відміну від класичного диференціального аналізу, для цього методу не існує формальної теоретичної моделі, яка дозволяє будувати аналітичні оцінки стійкості. Наявні методи пошуку неможливих диференціалів фактично виконують автоматичне конструювання шляхом перебору всіх можливих раундових диференціалів та знаходження несумісних шаблонів між ними. Втім, використання в алгоритмах шифрування нелінійних перетворень із максимально вирівняними диференціальними ймовірностями в цілому підвищує стійкість до даного виду аналізу.

1.2 Верхні межі ймовірностей існування нетривіальних диференціалів та диференціальних характеристик SP-мереж

SP-мережею будемо називати ітеративний блочний шифр виду:

$$E_k(X) = F_{K_r}(F_{K_{r-1}}(...(F_{K_1}(X)))),$$

де $K = (K_1,..,K_r) \in (V_u)^{mr}$ – ключ шифрування, а F раундове перетворення, яке визначається таким чином:

$$\begin{aligned} F : (V_u)^m \times (V_u)^m &\to (V_u)^m, \\ F(X,K) &= f(X \bullet K), \\ f(X) &= L(S(X)), \end{aligned}$$

де f – раундова функція, $S = (s_1,...,s_m)$ – рівень нелінійних підстановок, в якому

всі S блоки $S_i : V_u \to V_u$ – бієктивні, $L : (V_u)^m \to (V_u)^m$ лінійне перетворення (відносно операції $\circ$).

Основні результати щодо теоретичної стійкості SP-мереж викладені у серії робіт Хонга, Канга, Парка та ін. [6, 7, 5]. Одержані цими дослідниками оцінки справедливі лише для шифрів, марковських відносно операції $\oplus$. Зокрема, Хонг та ін. показали [7], що для дворaундової SP-мережі E, в якій перетворення L є лінійним відносно $\oplus$, задається у вигляді матриці та має максимальний індекс розгалуження $B = B(L) = m + 1$ або майже максимальний індекс розгалуження $B = B(L) = m$, має місце оцінка:

$$\forall x \forall a \forall b \neq 0 : d^E(x,a,b) \leq D^{B-1},$$

де $D = \max_i MDP(s_i)$.

Канг та ін. показали [6], що перетворення, яке задається у вигляді $(0,1)$-матриці, не може мати максимального індексу розгалуження. Також у [7] стверджувалось, що одержана Хонгом оцінка справедлива для перетворень із довільним індексом розгалуження, однак наведене доведення містить помилки.

Парк та ін. одержали більш точну оцінку стійкості для марковських SP-мереж за більш м'яких умов, ніж Хонг та ін. [5] Позначимо для дворaундової SP-мережі E

$$\begin{aligned} Q(s_i) &= \max\{\max_b \sum_a (d^{s_i}(a,b))^B, \max_a \sum_a (d^{s_i}(a,b))^B\}, \\ Q(E) &= \max_i Q(s_i). \end{aligned}$$

Тоді, якщо перетворення L із індексом розгалуження B є лінійним відносно $\oplus$, то справедлива оцінка:

$$\forall x \forall a \forall b \neq 0 : d^E(x,a,b) \leq Q(E).$$

Зокрема, оскільки виконується нерівність $Q(E) \leq D^{B-1}$, з оцінки Парка випливає оцінка Хонга для довільного значення індексу розгалуження. Зауважимо,

що, на відміну від результатів Хонга, для доведення оцінки Парка важлива лише лінійність L відносно $\oplus$; для інших операцій ця оцінка вже не справедлива. Також Парком було уточнено оцінки стійкості для AES-подібних шифрів; у [5] використовувалась доволі складна техніка доведення, однак в подальшому було знайдено більш просте доведення через каскадне представлення AES-подібного шифру. Користуючись результатами Парка, Келіхер у серії робіт [4] запропонував алгоритм обчислення ще більш точної оцінки верхньої межі для нетривіальних диференціалів багатораундових AES-подібних SP-мереж.

1.3 Приклад SP-мережі: алгоритм шифрування ДСТУ 7624:2014

Новий стандарт шифрування ДСТУ 7624:2014 [1] був розроблений колективом харківських криптографів на основі алгоритму шифрування «Калина» [3]. Він прийшов на заміну міждержавному стандарту ГОСТ 28147-89 (гармонізованому як ДСТУ ГОСТ 28147:2009 [2]).

Алгоритм ДСТУ 7624:2014 побудований на основі AES-подібної SP-мережі. Він перетворює блоки вхідного тексту довжиною 128, 256 або 512 біт у відповідні блоки шифртексту, за допомогою ключа довжиною 128, 256 або 512 біт. Розмір ключа може бути рівний розміру блоку, або може бути вдвічі більший за розмір блоку. Кількість раундів шифрування визначається довжиною ключа.

Вхідний блок даних розташовується у матриці стану; на кожному раунді шифрування над матрицею стану виконуються такі операції:

- додавання із раундовим ключем;
- нелінійна заміна ($SubBytes$);
- зсув рядків ($ShiftRows$);
- перемішування стовпчиків ($MixColumns$).

Після останнього раунду відбувається додаткове забілювання даних із окремим раундовим ключем. На першому раунді та після останнього раунду ключі додаються

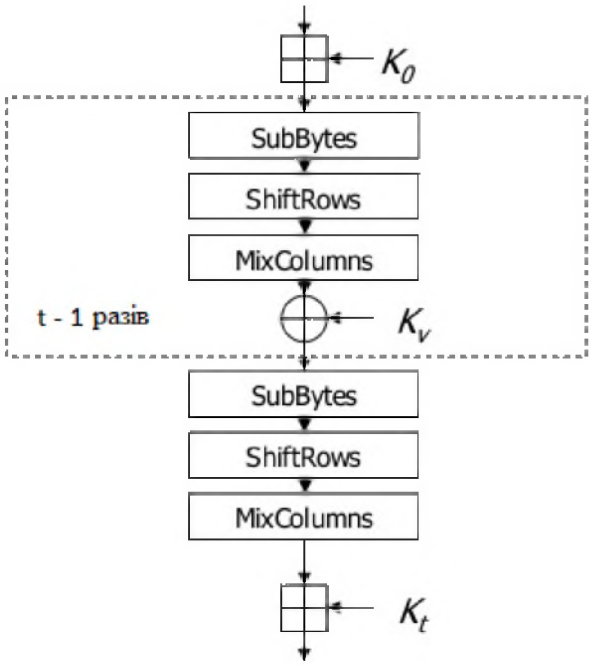

Рисунок 1.1 – Блок-схема алгоритму шифрування ДСТУ 7624:2014.

до матриці стану із використанням операції додавання за модулем 2^{64}, а в усіх інших раундах використовується побітове додавання. Це схематично зображено на рис. 1.1.

Нелінійна заміна байт матриці стану відбувається за допомогою чотирьох підстановок π_0, π_1, π_2, π_3 зафіксованих у стандарті. Перемішування стовпчиків відбувається шляхом множення на циркулянтну матрицю над полем $GF(2^8)$; це перетворення має максимально можливий індекс розгалуження $B = 9$.

1.4 Оцінки стійкості алгоритму шифрування ДСТУ 7624:2014

В роботі [12] було наведено такі твердження:

Якщо ζ являється шифром ДСТУ 7624:2014 з випадковими та рівноймовірними раундовими ключами, з довжиною блоку n біт та r раундами шифрування. Тоді при $r \geq 6$ для будь-якої бінарної операції $\circ$ на множині V_n виконується нерівність:

$$\max_{\alpha,\beta\in V_n\setminus\{0\}}\{d^{(\zeta_{[1,r-1]})}(\alpha,\beta)\} \le \begin{cases} 2^{-80}, & \text{якщо } n=128; \\ 2^{-160}, & \text{якщо } n=256; \\ 2^{-320}, & \text{якщо } n=512. \end{cases}$$

де $d^s(a,b)$ імовірність диференціалу (a,b) для S-блоку s.

$$QD(s,B)=\max_{a,b\neq 0}\{\sum_c (d^s(a,c))^B, \sum_c (d^s(c,b))^B\},$$

У роботі Яковлєва [13] було наведено обчислення величин QD для імовірностей диференціалів та лінійних апроксимацій та величин QD^* (QD^* – уточнена оцінка QD) для імовірностей диференціалів S-блоків алгоритму шифрування ДСТУ 7624:2014. Результати отримані, Яковлєвим, наведені у таб. 1.1.

Таблиця 1.1 – Значення параметрів QD та QD^* S-блоків шифру ДСТУ 7624:2014.

S-блок	QD	QD^*
π_0	$2^{-43,98}$	$2^{-43,996}$
π_1	$2^{-43,93}$	$2^{-44,169}$
π_2	$2^{-44,69}$	$2^{-44,823}$
π_3	$2^{-44,85}$	$2^{-44,882}$

Висновки до розділу 1

В даному розділі проведено аналіз опублікованих результатів, відбір та узагальнення теоретичних відомостей, необхідних для подальшого дослідження; також в розділі викладено принципи побудови диференціальної атаки на

ітеративний шифр. На основі проведеного аналізу встановлено, що існуючий математичний апарат дозволяє адекватно оцінювати стійкість марковських симетричних блочних шифрів. Визначення теоретичної та практичної стійкості SP-мереж вимагає знаходження нових, та суттєвої переробки існуючих підходів до створення відповідних математичних моделей та методів їх оцінювання, в залежності від конкретної структури шифруючого перетворення.

2 ФОРМАЛІЗОВАНИЙ МЕТОД ОЦІНКИ СТІЙКОСТІ SP-МЕРЕЖ ДО ДИФЕРЕНЦІАЛЬНОГО КРИПТОАНАЛІЗУ

Даний розділ присвячено дослідженню теоретичної та практичної стійкості SP-мереж до диференціального криптоаналізу, зокрема побудові алгоритмів автоматичного оцінювання верхніх меж диференціальних імовірностей та диференціальних характеристик для SP-мереж. Враховуючи особливості структури SP-мереж, в даному розділі розглядаються диференціали за операцією $\oplus$.

У 2001-2003 роках Л. Келіхер разом із В. Мейєром та С. Таваресом розробив декілька складних, але елегантних методик обчислення параметрів стійкості шифру AES до лінійного криптоаналізу, що одержали назву алгоритмів KMT1 та KMT2 відповідно. Згодом Келіхер переніс цю методику на диференціальний криптоаналіз, а потім адаптував її для обчислення параметрів стійкості алгоритму шифрування Camellia, який побудовано на основі схеми Фейстеля.

Алгоритми Келіхера для розрахунків використовували відомості про повний розподіл диференціальних імовірностей S-блоків та шляхи проходження активних байтів через лінійні перетворення. Розрахунки Келіхера дозволили уточнити верхні оцінки для імовірностей диференціалів для вказаних шифрів: 2^{-111} для AES (проти аналітичного значення 2^{-96}) та 2^{-55} для Camellia (проти аналітичного значення 2^{-28}).

Слід зазначити, що методики Келіхера застосовні лише для алгоритмів шифрування, що є марковськими відносно операції $\oplus$, і при цьому вони переускладнені для досягання більшої ефективності обрахунків. В даному розділі наводяться більш прості методики визначення верхніх меж імовірностей диференціалів, опубліковані в [15] та уточнені мною, що застосовні для довільних немарковських блокових шифрів, побудованих із використанням S-блоків на основі структур SP-мереж.

2.1 Алгоритми побудови оцінок стійкості немарковських SP-мереж до диференціального криптоаналізу

В основі алгоритмів лежить ідея Келіхера [4], а саме: заміна точних значень різниць α, β на так звані «шаблони». Для заданого $\alpha \in V_n \equiv (V_u)^m$ визначимо шаблон $T\alpha = \widehat{\alpha} \in V_m$ за таким правилом: $\widehat{\alpha}_i = 0$, якщо $\alpha_i = 0$ та $\widehat{\alpha}_i = 1$, якщо $\alpha_i \neq 0$. Використовуючи шаблони, ми можемо побудувати для кожного r матрицю верхніх меж імовірностей диференціалів ($UB^{[r]}[\widehat{\alpha},\widehat{\beta}]$) таку, щоб для довільних векторів α, β виконувалась рівність:

$$d(\alpha,\beta)^{[r]} \leq UB^{[r]}[\widehat{\alpha},\widehat{\beta}],$$

під $d(\alpha,\beta)^{[r]}$ маємо на увазі імовірність r-раундового диференціалу $(\alpha\beta)$.

Нехай L – лінійне перетворення алгоритму шифрування. Для того, щоб встановити кількість векторів γ, які відповідають таким умовам: $T\gamma = \widehat{\gamma}$ та $T(L(\gamma)) = \widehat{\beta}$, вводимо допоміжну матрицю $W[\widehat{\alpha},\widehat{\beta}]$:

$$W[\widehat{\alpha},\widehat{\beta}] = \left| \left\{ x \in V_n : Tx = \widehat{\alpha}, T(L(x)) = \widehat{\beta} \right\} \right|.$$

Матриця W показує кількість способів отримати із заданого вхідного шаблону $\widehat{\alpha}$ заданий вихідний шаблон $\widehat{\beta}$ для лінійного перетворення L. Таким чином, маємо наступну оцінку:

$$d(\alpha,\beta)^{[r]} \leq \sum_{\widehat{\gamma}} UB^{[r-1]}[\widehat{\alpha},\widehat{\gamma}] \cdot W[\widehat{\gamma},\widehat{\beta}] \cdot p^{wt(\widehat{\beta})},$$

де $p = \max(MDP(s_i))$.

Підсумуючи наведене вище, наведемо два алгоритми обчислення верхніх меж диференціальних імовірностей для немарковських SP-мереж які були наведені в [15]. Ці алгоритми виступають аналогами алгоритмів КМТ1 та КМТ2.

Алгоритм 2.1. Обчислення верхніх меж імовірностей диференціалів SP-мережі без використання точних значень розподілів диференціальних імовірностей S-блоків.

Вхід: SP-мережа E, кількість раундів r.

Вихід: матриця верхніх меж $UB^{[r]}[*,*]$.

Передобчислення: матриця $W[*,*]$, число $p = \max(MDP(s_i))$.

а) Для всіх шаблонів $(\widehat{\alpha},\widehat{\beta})$ встановити $UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{p^{B-1},p^{wt(\widehat{\beta})}\right\}$.

б) Для $t = 3,4,...r$ виконати такі кроки:

1) Для всіх шаблонів $\widehat{\beta}$ виконати такі кроки:

Встановити:

$$M = \min_{\widehat{\gamma}} UB^{[t-1]}[\widehat{\gamma},\widehat{\beta}].$$

Для всіх шаблонів $\widehat{\alpha}$ встановити:

$$UB^{[t]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{M, \sum_{\widehat{\gamma}} UB^{[t-1]}[\widehat{\alpha},\widehat{\gamma}] \cdot W[\widehat{\gamma},\widehat{\beta}] \cdot p^{wt(\widehat{\beta})}\right\}.$$

в) Повернути матрицю $UB^{[r]}[*,*]$.

Алгоритм 2.2. Обчислення верхніх меж імовірностей диференціалів SP-мережі із використанням точних значень розподілів диференціальних імовірностей S-блоків.

Вхід: SP-мережа E, кількість раундів r.

Вихід: матриця верхніх меж $UB^{[r]}[*,*]$.

Передобчислення: матриця $W[*,*]$, число $p = \max(MDP(s_i))$, розподіли $u_i(A)(A = \overline{1,m})$.

а) Для всіх шаблонів $\widehat{\alpha},\widehat{\beta}$ встановити:

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{\sum_{i=1}^{W[\widehat{\alpha},\widehat{\beta}]} u_i(wt(\widehat{\alpha})) \cdot u_i(wt(\widehat{\beta})), p^{wt(\widehat{\beta})}\right\}.$$

б) Для $t = 3,4,...r$ виконати такі кроки:

1) Для всіх шаблонів $\widehat{\beta}$ виконати такі кроки:

Встановити:

$$M = \min_{\widehat{\gamma}} UB^{[t-1]}[\widehat{\gamma},\widehat{\beta}].$$

Для всіх шаблонів $\widehat{\alpha}$ встановити:

$$UB^{[t]}[\widehat{\alpha},\widehat{\beta}] = \min \left\{ M, \sum_{\widehat{\gamma}:W[\widehat{\gamma},\widehat{\beta}]>0} UB^{[t-1]}[\widehat{\alpha},\widehat{\gamma}] \cdot \sum_{i=1}^{W[\widehat{\gamma},\widehat{\beta}]} u_i(wt(\widehat{\beta})) \right\}.$$

в) Повернути матрицю $UB^{[r]}[*,*]$.

2.1.1 Передобчислення матриці W

Нагадаємо, що, за визначенням, елементи матриці $W[\widehat{\alpha},\widehat{\beta}]$ дорівнюють:

$$W[\widehat{\alpha},\widehat{\beta}] = \left| \left\{ x \in V_n : Tx = \widehat{\alpha}, T(L(x)) = \widehat{\beta} \right\} \right|.$$

Найпростіший спосіб знаходження $W[\widehat{\alpha},\widehat{\beta}]$ – це безпосередній перебір всіх можливих x та обчислення відповідних шаблонів від входу та виходу. Цей метод безвідмовно працює для невеликих довжин вхідних векторів, але вже для $n = 64$ він стає обчислювально невиконуваним. Також, безпосередній перебір може стати в нагоді, коли лінійне перетворення розпадається на декілька незалежних відображень.

Лінійне перетворення $L : V_n \to V_n$ задано матрицею L над V_u. Через матрицю $L_{\widehat{\alpha},\widehat{\beta}}$ позначимо таку підматрицю матриці L, яка розташована на перетині тих стовпчиків, для яких $\widehat{\alpha}_i = 1$, та тих рядків, для яких $\widehat{\beta}_i = 0$. Через $W_{\geq}[\widehat{\alpha},\widehat{\beta}]$ позначимо кількість розв'язків рівняння $L_{\widehat{\alpha},\widehat{\beta}} \cdot z = 0$. Звідси маємо:

$$W_{\geq}[\widehat{\alpha},\widehat{\beta}] = (2^u)^{wt(\widehat{\alpha}) - rank(L_{\widehat{\alpha},\widehat{\beta}})}.$$

Якщо лінійне перетворення задається MDS-матрицею, можна значно спростити підрахунок $W_{\succeq}[\widehat{\alpha},\widehat{\beta}]$. Так як довільна підматриця MDS-матриці є невиродженою, то ранг будь-якої довільної підматриці MDS-матриці буде рівний $rank(L_{\widehat{\alpha},\widehat{\beta}}) = min(wt(\widehat{\alpha}),wt(\widehat{\beta}))$. Звідси випливає рівність:

$$W_{\succeq}[\widehat{\alpha},\widehat{\beta}] = (2^u)^{wt(\widehat{\alpha})-min(wt(\widehat{\alpha}),wt(\widehat{\beta}))}.$$

Із значень $W_{\succeq}[\widehat{\alpha},\widehat{\beta}]$ знаходяться значення $W[\widehat{\alpha},\widehat{\beta}]$ за допомогою формул включення-виключення:

$$W'[\widehat{\alpha},\widehat{\beta}] = \sum_{k=0}^{wt(\widehat{\alpha})} \sum_{\substack{\widehat{c}:\widehat{c}\prec\widehat{\alpha}, \\ wt(\widehat{c})=k}} (-1)^{wt(\widehat{\alpha})-k} \cdot W_{\succeq}[\widehat{\alpha},\widehat{c}],$$

$$W[\widehat{\alpha},\widehat{\beta}] = \sum_{k=0}^{wt(\widehat{\beta})} \sum_{\substack{\widehat{c}:\widehat{c}\prec\widehat{\beta}, \\ wt(\widehat{c})=k}} (-1)^{wt(\widehat{\beta})-k} \cdot W'[\widehat{c},\widehat{\beta}],$$

де символом $\prec$ позначено відношення домінування на бітових векторах.

2.1.2 Обчислення границь розподілів імовірностей диференціалів комбінацій активних S-блоків

Нехай $u_i(A)(A = \overline{1,m})$ – послідовності верхніх меж ненульових імовірностей диференціалів комбінацій з A активних S-блоків, які відсортовані у порядку спадання. Ці послідовності обчислюються як A-мірна згортка відсортованих розподілів імовірностей диференціалів задіяних S-блоків.

Для деякого зафіксованого $\alpha \in V_u$ розглянемо всі $\beta \in V_u$. Пронумеруємо вектори $\beta \neq 0$ так, щоб послідовність $d_{\alpha,i} = d^{S_k}(\alpha,\beta_i)$ була незростаючою. Таким чином $u_i(1) = \max_{\alpha\neq 0} d_{\alpha,i}$. Тоді послідовність $u_i(1)$ є незростаючою.

Послідовність $u_i(A)$ обчислюється за індукцією:

а) обчислюється згортка послідовностей $(v_i(A)) = (u_i(A-1)) \oplus (u_i(1))$;

б) $(u_i(A))$ – відсортована послідовність $(v_i(A))$.

Послідовність $u_i(A)$ можна подати за допомогою списків $\{p_j(A),N_j(A)\}$, де p_j – значення імовірностей $u_i(A)$, а $N_j(A)$ кількість входжень значень p_j у $u_i(A)$. Тоді обчислити згортку $(u_i(A-1)) \oplus (u_i(1))$ можна так:

а) обчислюються всі можливі значення добутків $p_l(A-1) \cdot p_t(1)$;

б) значення добутків відсортовуються та залишаємо тільки неповторювані у послідовність $p_j(A)$;

в) для одержаних значень послідовності $p_j(A)$ обчислюємо:

$$N_j(A) = \sum_{l,t} N_l(A-1) \cdot N_t(1),$$

де l,t такі, що $p_j(A) = p_l(A-1)p_t(1)$.

Алгоритм 2.3. Обчислення $\sum_{i=1}^{W[\widehat{\alpha},\widehat{\beta}]} u_i(wt(\widehat{\alpha})) \cdot u_i(wt(\widehat{\beta}))$.

Вхід: значення $\widehat{\alpha}$,$\widehat{\beta}$.

Вихід: значення суми.

Передобчислення: значення $W[\widehat{\alpha},\widehat{\beta}]$, розподіли $u_i(wt(\widehat{\alpha}))$ та $u_i(wt(\widehat{\beta}))$ подані в вигляді списків $\{p_i(A),N_i(A)\}$ та $\{p_j(A),N_j(A)\}$ відповідно.

$sum := 0, W := W[\widehat{\alpha},\widehat{\beta}], i := 1, j := 1$

а) $m = \min\{N_i,N_j\}$

б) Якщо $m \le W$ тоді:

1) $sum := sum + p_i \cdot p_j \cdot m$;

2) якщо $N_i \le N_j$ тоді:

$i := i + 1$;

$N_j := N_j - m$;

інакшке:

$j := j + 1$;

$N_i := N_i - m$;

в) інакше:

1) $sum := sum + p_i \cdot p_j \cdot W$;

2) повернути sum;

г) $W := W - m$.

2.1.3 Властивості матриці UB

Слід зазначити, що метою обчислення матриці верхніх меж імовірностей диференціалів SP-мережі є не тільки отримання аналітичних результатів щодо теоретичної (доказової) стійкості до диференціального криптоаналізу. Так як значення матриці визначають верхні оцінки меж імовірностей диференціалів, то там, де диференціал рівний нулю – диференціал неможливий. Тобто кількість нулів в матрці $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ дає можливість провести аналіз неможливих диференціалів. Аналіз неможливих диференціалів являється альтернативною версією класичного диференціального криптоаналізу.

Також матриця $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ має такі властивості:

1) $UB^{[t]}[0,0] = 1$;

2) перший рядок рівний 0 (крім елемента $UB^{[t]}[0,0]$);

3) перший стовпчик рівний 0 (крім елемента $UB^{[t]}[0,0]$).

2.2 Особливості застосування алгоритмів до алгоритму шифрування ДСТУ 7624:2014

Ми розглядаємо алгоритм шифрування ДСТУ 7624:2014 з розміром блоку 128 біт. Для ДСТУ 7624:2014 перетворення $MixColumns$ задається матрицею розмірності 8×8. Проте лінійне перетворення перетворює 16 бітні вектори в 16 бітні вектори, тому для обрахунку вводиться нова матриця $W_{full}[\widehat{\alpha},\widehat{\beta}]$. $W_{full}[\widehat{\alpha},\widehat{\beta}]$ обраховується з

передобчисленої матриці W:

$$W_{full}[\widehat{\alpha},\widehat{\beta}] = W[\widehat{c_1},\widehat{\beta_1}] \cdot W[\widehat{c_2},\widehat{\beta_2}],$$

де $\widehat{c} = ShiftRows(\widehat{\alpha})$, а індекси 1 та 2 відповідають першим 8 бітам та наступним 8 бітам вектору відповідно.

ДСТУ 7624:2014 відноситься до класу марківських SP-мереж тому ми уможемо уточнити формулу для обчислення матриці меж імовірностей двораундових диференціалів алгоритму обчислення верхніх меж імовірностей диференціалів SP-мережі без використання точних значень розподілів диференціальних імовірностей S-блоків. Матриця $UB^{[2]}[\widehat{\alpha},\widehat{\beta}]$ буде мати вигляд:

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \begin{cases} \min\left\{p^{B-1}, p^{wt(\widehat{\alpha})}, p^{wt(\widehat{\beta})}\right\}, & wt(\widehat{\alpha}) + wt(\widehat{\beta}) \geq B \\ 0, & \text{в інших випадках,} \end{cases}$$

а матриця $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ буде мати вигляд:

$$UB^{[t]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{M, \sum_{\widehat{\gamma}} UB^{[t-1]}[\widehat{\alpha},\widehat{\gamma}] \cdot W_{full}[\widehat{\gamma},\widehat{\beta}] \cdot p^{wt(\widehat{\beta})}\right\}.$$

Для алгоритму обчислення верхніх меж імовірностей диференціалів SP-мережі із використанням точних значень розподілів диференціальних імовірностей S-блоків мариця $UB^{[2]}[\widehat{\alpha},\widehat{\beta}]$ буде мати вигляд:

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{\sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(wt(\widehat{\alpha})) \cdot u_i(wt(\widehat{\beta})), p^{wt(\widehat{\alpha})}, p^{wt(\widehat{\beta})}, p^{B-1}\right\},$$

а матриця $UB^{[t]}(\widehat{\alpha},\widehat{\beta})$ буде мати вигляд:

$$UB^{[t]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{M, \sum_{\widehat{\gamma}:W_{full}[\widehat{\gamma},\widehat{\beta}]>0} UB^{[t-1]}[\widehat{\alpha},\widehat{\gamma}] \cdot \sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(wt(\widehat{\beta}))\right\}.$$

Також оцінка може бути покращена якщо замість p^B брати QD (значення QD наведені у таб. 1.1). Тобто:

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \begin{cases} \min\left\{p^{B-1}, QD, p^{wt(\widehat{\beta})}\right\}, & wt(\widehat{\alpha}) + wt(\widehat{\beta}) \geq B \\ 0, & \text{в інших випадках,} \end{cases}$$

для алгоритму 2.1,

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{\sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(wt(\widehat{\alpha})) \cdot u_i(wt(\widehat{\beta})), p^{wt(\widehat{\alpha})}, p^{wt(\widehat{\beta})}, p^{QD}\right\},$$

для алгоритму 2.2.

Як відомо нелінійна заміна байт матриці стану відбувається за допомогою чотирьох підстановок зафіксованих у стандарті π_0, π_1, π_2, π_3, тому обчислення розподілів $u_i(A)$ буде виглядати дещо інакше. Як уже описувалось вище, ми працюємо з 16-бітними векторами.

1) Вхідний вектор розбивається на чотири 4-бітні ветори;

2) кожному біту з цих чотирьох векторів ставиться в відповідність підстановка (першому – π_0, другому – π_1, третьому – π_2, четвертому – π_3);

3) для кожного ненульового біта обчислюється розподіл $u_i(A)$. Загальний розподіл буде виглядати як згортка обчислених раніше $u_i(A)$.

Тобто формула для матриці $UB^{[2]}[\widehat{\alpha},\widehat{\beta}]$ буде мати вигляд:

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{\sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(\widehat{\alpha}) \cdot u_i(\widehat{\beta}), p^{wt(\widehat{\alpha})}, p^{wt(\widehat{\beta})}\right\},$$

для матриці $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ аналогічно:

$$UB^{[t]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{M, \sum_{\widehat{\gamma}:W_{full}[\widehat{\gamma},\widehat{\beta}]>0} UB^{[t-1]}[\widehat{\alpha},\widehat{\gamma}] \cdot \sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(\widehat{\beta})\right\}.$$

2.3 Оцінка складності та труднощі в реалізації алгоритмів

В даному підрозділі будуть описані оцінки складності підрахунків та передобчислень даних алгоритмів та описані складності в розрахунку даних алгоритмів при застосуванні їх до шифру ДСТУ 7624:2014 з розміром блоку 128 біт.

2.3.1 Передобчислення матриці W

Обчислення значень $W_{\geq}[\widehat{\alpha},\widehat{\beta}]$ в загальному випадку зводиться до обчислення всіх підматриць матриці $L_{\widehat{\alpha},\widehat{\beta}}$. Всього $L_{\widehat{\alpha},\widehat{\beta}}$ – 2^{2m}. Складність знаходження рангу матриці методом Гауса оскладає $O(m^3)$ операцій віднімання дробових чисел. Таким чином, складність обчислення занчень $W_{\geq}[\widehat{\alpha},\widehat{\beta}]$ – $O(m^3 \cdot 2^{2m})$.

Якщо в основі лінійного перетворення розглядуваного алгоритму шифрування лежить MDS-матриця, то обчислення всіх значень $W_{\geq}[\widehat{\alpha},\widehat{\beta}]$ займає $O(4^m)$ операцій побітового зсуву. Обчислення значень $W_{\geq}[\widehat{\alpha},\widehat{\beta}]$ може бути розпаралелене за параметром $\widehat{\alpha}$.

Для передобчислення $W[\widehat{\alpha},\widehat{\beta}]$, також необхідно перебрати 2^{2m} значень, складність обчислень $O(2^m)$ операцій додавання цілих чисел. Обчислення значень $W[\widehat{\alpha},\widehat{\beta}]$ може бути розпаралелене за параметром $\widehat{\alpha}$.

Складність передобчислення послідовності верхніх меж ненульових імовірностей диференціалів $u_i(A)$ є відносно невеликою. Труднощі, які можуть виникнути в ході

обрахунку, це – похибки округлення та/або переповнення.

2.3.2 Обчислення матриці UB

Для підрахунку матриці верхніх меж імовірностей t-раундових диференціалів $UB^{[t]}(\widehat{\alpha},\widehat{\beta})$ необхідно перебрати всі вектори $\widehat{\alpha}$ та $\widehat{\beta}$ (це $2^{16} \times 2^{16}$ значень). Також необхідно мати обраховану матрицю $UB^{[t-1]}(\widehat{\alpha},\widehat{\beta})$. Складність алгоритму (якщо ми маємо обраховану $UB^{[t-1]}(\widehat{\alpha},\widehat{\beta})$) – $O(2^{16})$ операцій множення дробових чисел. Значення матриці W_{full} (порядку близько 2^{128}) значно більші за стандартні типи данних ЕОМ.

Для зберігання $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ необхідні значні ресурси памяті (32 GB на одну матрицю, а таких матриць 8). Оцінка складності обрахунку однієї матриці – $O(2^{42})$ операцій множення дробових чисел.

Висновки до розділу 2

В даному розділі були наведені та уточнені алгоритми обчислення верхніх меж імовірностей диференціалів для SP-мереж. Алгоритми, що пропонуються, враховують особливості структури блокового шифру, характеристики лінійних перетворень, які використовуються під час шифрування, та граничні значення для розподілів імовірностей диференціалів окремих S-блоків шифру.

Складність алгоритмів у більшості випадків є достатньою для реалізації на персональних комп'ютерах. Очікується, що пропоновані алгоритми дозволять для конкретних шифрів підсилити відомі аналітичні результати щодо теоретичної (доказової) стійкості до диференціального криптоаналізу, отримати кількості неможливих диференціалів, що в свою чергу, характеризує стійкість до аналізу

неможливих диференціалів.

3 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

В даному розділі будуть наведені результати експериментальних досліджень, а саме: застосування уточненого алгоритму 2.1 та уточненого алгоритму 2.2 до модифікованого (спрощеного) шифру ДСТУ 7624:2014 та до алгоритму шифрування ДСТУ 7624:2014 з розміром блоку 128 біт. В якості результата очікуємо оцінку верхніх меж імовірностей диференціалів та кількості нулів в матриці верхніх меж імовірностей диференеціалів ($UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$).

3.1 Передобчислення матриці W

Так як в лінійне перетворення ДСТУ 7624:2014 задається MDS-матрицею, передобчислення матриці W виконувалось за такими формулами:

$$
\begin{aligned}
W_{\geq}[\widehat{\alpha},\widehat{\beta}] &= (2^u)^{wt(\widehat{\alpha})-min(wt(\widehat{\alpha}),wt(\widehat{\beta}))}, \\
W'[\widehat{\alpha},\widehat{\beta}] &= \sum_{k=0}^{wt(\widehat{\alpha})} \sum_{\substack{\widehat{c}:\widehat{c}\prec\widehat{\alpha}, \\ wt(\widehat{c})=k}} (-1)^{wt(\widehat{\alpha})-k} \cdot W_{\geq}[\widehat{\alpha},\widehat{c}], \\
W[\widehat{\alpha},\widehat{\beta}] &= \sum_{k=0}^{wt(\widehat{\beta})} \sum_{\substack{\widehat{c}:\widehat{c}\prec\widehat{\beta}, \\ wt(\widehat{c})=k}} (-1)^{wt(\widehat{\beta})-k} \cdot W'[\widehat{c},\widehat{\beta}].
\end{aligned}
$$

Результуюча матриця виявилась розрідженою. Діаграма відносної кількості нульових елементів до ненульових елементів показана на рис. 3.1.

Ця властивість значно спрощує обрахунок матриць верхніх меж імовірностей диференеціалів. Ми можемо винести в передобчислення ті α і β, для яких $W[\alpha,\beta] \neq 0$.

Таких значень α і β – 40%. Тим самим ми пришвидшуемо роботу алгоритмів

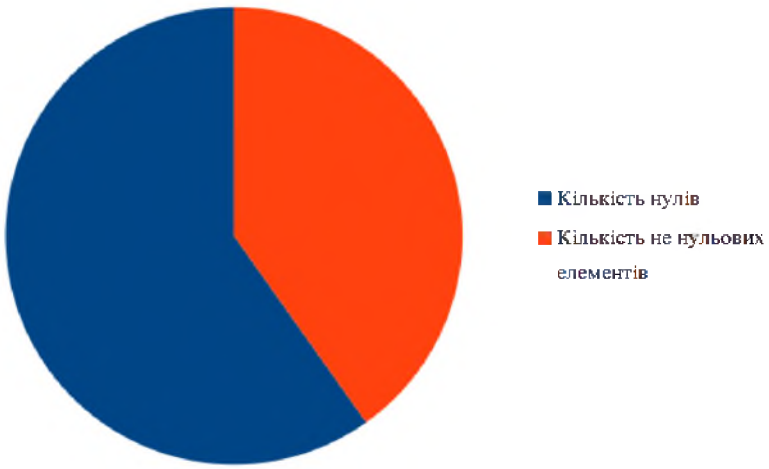

Рисунок 3.1 – Діаграма відносної кількості нульових елементів до не нульових.

обрахунку верхніх меж імовірностей диференціалів на 60%. Ці результати справедливі для будь-якої SP-мережі, лінійне перетворення якої задається MDS-матрицею 8×8.

3.2 Оцінювання стійкості модифікованого (спрощеного) шифру ДСТУ 7624:2014

Для попереднього дослідження була обрана модифікація алгоритму шифрування ДСТУ 7624:2014. Розмір блоку був зменшений до 64 біт. Лінійне перетворення задається MDS-матрицею 4×4. Раунд шифрування не змінився:

- додавання із раундовим ключем;
- нелінійна заміна ($SubBytes$);
- зсув рядків ($ShiftRows$);
- перемішування стовпчиків ($MixColumns$).

Зсув рядків ($ShiftRows$) зсуває відповідні блоки вдвічі меньшого розміру. Підстановки π_0, π_1, π_2, π_3 залишелись без змін.

Для обрахунку був обраний алг. 2.2. з уточненими формулами тобто:

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{\sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(\widehat{\alpha})\cdot u_i(\widehat{\beta}), p^{wt(\widehat{\alpha})}, p^{wt(\widehat{\beta})}, p^{B-1}\right\}, \quad (3.1)$$

$$UB^{[t]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{M, \sum_{\widehat{\gamma}:W_{full}[\widehat{\gamma},\widehat{\beta}]>0} UB^{[t-1]}[\widehat{\alpha},\widehat{\gamma}]\cdot \sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(\widehat{\beta})\right\}. \quad (3.2)$$

Також були проведені розрахунки за такими формулами:

$$UB^{[2]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{\sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(\widehat{\alpha})\cdot u_i(\widehat{\beta}), p^{wt(\widehat{\alpha})}, p^{wt(\widehat{\beta})}, p^{QD}\right\}, \quad (3.3)$$

$$UB^{[t]}[\widehat{\alpha},\widehat{\beta}] = \min\left\{M, \sum_{\widehat{\gamma}:W_{full}[\widehat{\gamma},\widehat{\beta}]>0} UB^{[t-1]}[\widehat{\alpha},\widehat{\gamma}]\cdot \sum_{i=1}^{W_{full}[\widehat{\alpha},\widehat{\beta}]} u_i(\widehat{\beta})\right\}. \quad (3.4)$$

Для обчислення $\sum_{i=1}^{W[\widehat{\alpha},\widehat{\beta}]} u_i(wt(\widehat{\alpha}))\cdot u_i(wt(\widehat{\beta}))$ використувався Алгоритм 2.3.

Були отримані такі значення верхніх меж в залежності від номеру раунду (для формули 3.1), результати наведені в таб. 3.1. Для формули (3.3) в таб. 3.2. Також було підраховано кількість нульових диференціалів, результати наведені в таб. 3.3.

Таблиця 3.1 – Значення верхніх меж в залежності від номеру раунду (формула 3.1).

Номер раунду	Значення верхньої межі імовірності диференціалів
2	2^{-20}
3	$2^{-33.1688}$
4	$2^{-33.1688}$
5	$2^{-33.1688}$
6	$2^{-33.1688}$

Продовження таблиці 3.1

Номер раунду	Значення верхньої межі імовірності диференціалів
7	$2^{-33.1688}$
8	$2^{-33.1688}$
9	$2^{-33.1688}$

Таблиця 3.2 – Значення верхніх меж в залежності від номеру раунду (формула 3.3).

Номер раунду	Значення верхньої межі імовірності диференціалів
2	$2^{-23.6131}$
3	$2^{-33.1688}$
4	$2^{-33.1688}$
5	$2^{-33.1688}$
6	$2^{-33.1688}$
7	$2^{-33.1688}$
8	$2^{-33.1688}$
9	$2^{-33.1688}$

Таблиця 3.3 – Кількість нульових диференціалів в залежності від номеру раунду.

Номер раунду	Кількість диференціалів
2	56190
3	12800
4	900
5	0
6	0
7	0
8	0
9	0

З результатів наведених у таб. 3.2 можна зробити висновок, що починаючи з 5 раунду нульові диференціали відсутні. Тобто модифікований алгоритм шифрування ДСТУ 7624:2014 є стійким до аналізу неможливих диференціалів. З результатів наведених у таб. 3.1 можна зробити висновик, що алгоритм 2.2 шифрування дає грубіші оцінки ніж очікувалось.

3.3 Оцінювання стійкості шифру ДСТУ 7624:2014 з розміром блоку 128 біт

Обрахунки проводились за допомогою алг. 2.1. з уточненими формулами тобто:

$$UB^{[2]}(\widehat{\alpha},\widehat{\beta}) = \begin{cases} \min\{p^{B-1},p^{wt(\widehat{\alpha})},p^{wt(\widehat{\beta})}\}, & wt(\widehat{\alpha})+wt(\widehat{\beta}) \geq B \\ 0, & \text{в інших випадках,} \end{cases}$$

$$UB^{[t]}(\widehat{\alpha},\widehat{\beta}) = \min\left\{M, \sum_{\widehat{\gamma}} UB^{[t-1]}(\widehat{\alpha},\widehat{\gamma}) \cdot W_{full}[\widehat{\gamma},\widehat{\beta}] \cdot p^{wt(\widehat{\beta})}\right\}.$$

Використовуючи властивість розрідженості матриці W для кожного α були обраховані всі β для яких $W_{full}[\alpha,\beta] \neq 0$.

Слід зазначити, що це передобчислення займає 1,7 GB памяті.

Також підрахунок $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ був розпаралелений за параметром $\widehat{\alpha}$.

Обрахунок однієї матриці $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ займав близько 20 годин на процесорі Intel® Core™ i5-3230M CPU @ 2.60GHz × 4 під операційною системою Ubuntu 15.10 з компілятором GCC версії 5.3.1.

Отримана оцінка межі імовірностей диференціалів – 2^{-40}. Також було підраховано кількість нульових елементів матриці $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$.

Графік кількості нулів в залежності від матриці $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$ наведений на рис. 3.2.

З графіка наведеного на рис. 3.1 можна зробити висновок, що починаючи з 4 раунду нульові диференціали відсутні. Тобто алгоритм шифрування ДСТУ 7624:2014 є стійким до аналізу неможливих диференціалів. З отриманої оцінки верхніх меж імовірностей диференціалів можна зробити висновок, що алгоритм 2.1 дає грубіші оцінки ніж очікувалось.

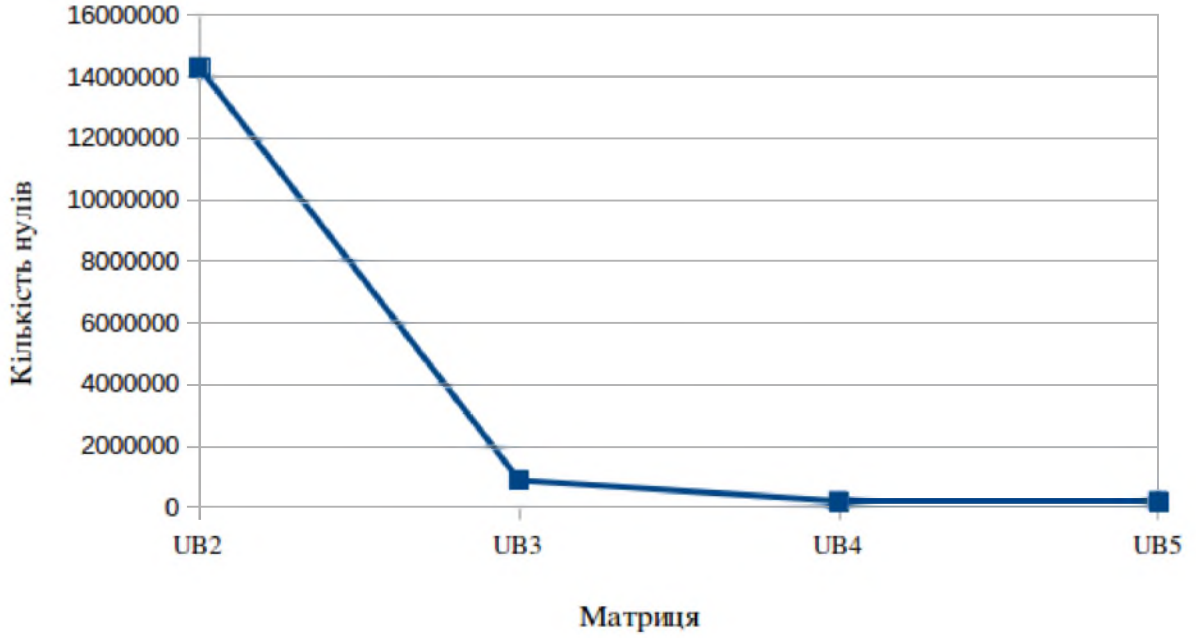

Рисунок 3.2 – Графік кількості нулів в залежності від матриці $UB^{[t]}[\widehat{\alpha},\widehat{\beta}]$.

Висновки до розділу 3

В цьому розділі наведені основні результати роботи, а саме верхні межі імовірностей диференеціалів та кількості нульових диференціалів для різних раундів. Висновком з цих оцінок є те, що уточнені алгоритми дають грубу оцінку ($2^{-33.1688}$ для модифікованого шифру ДСТУ 7624:2014 та 2^{-40} для ДСТУ 7624:2014 з розіром блоку 128-біт). Нульові диференціали відсутні починаючи з 5 раунду.

ВИСНОВКИ

В результаті виконання даної роботи було проаналізовано структуру SP-мережі, як приклад було розглянуто алгоритм шифрування ДСТУ 7624:2014.

Було досліджено методи побудови матриць верхніх меж імовірностей диференеціалів для SP-мереж.

Було уточнено методи побудови матриць верхніх меж імовірностей диференеціалів для маркіських SP-мереж.

Програмно реалізоано алгоритми для знаходження верхніх меж імовірностей диференеціалів для маркіських SP-мереж. Данний алгоритм було використано для знаходження верхніх меж імовірностей диференеціалів шифру ДСТУ 7624:2014 та його модифікації.

Отримані результати свічать про те, що уточнені алгоритми дають грубу оцінку ($2^{-33.1688}$ для модифікованого шифру ДСТУ 7624:2014 та 2^{-40} для ДСТУ 7624:2014 з розіром блоку 128-біт).

ПЕРЕЛІК ПОСИЛАНЬ

1. Інформаційні технології. Криптографічний захист інфомації. Алгоритм симетричного блокового перетворення: ДСТУ 7624:2014. — Держспоживстандарт України, 2014. — 238 с.

2. Системы обработки информации. Защита криптографическая. Алгоритмы криптографического преобразования: ДСТУ ГОСТ 28147:2009. — Держспоживстандарт України, 2008. — 28 с.

3. Горбенко І. Д. Перспективний блоковий симетричний шифр «калина» – основні положення та специфікація. / І. Д. Горбенко В. І. Долгов, Р. В. Олійников та ін. // Прикладная радиоэлектроника. — 2007. — С. 195–208.

4. Keliher L. Toward provable security against differential and linear cryptanalysis for camellia and related ciphers / L. Keliher. Режим доступу — http://ijns.femto.com.tw/contents/ijns-v5-n2/ijns-2007-v5-n2-p167-175.pdf.

5. Park S. On the security of rijndael-like structures against differential and linear cryptanalysis / S. Park, S.H. Sung, S. Chee, E.J. Yoon, J. Lim // Advances in Cryptology, ASIACRYPT 2002. — 2002. — Vol. 2501. — P. 176–191.

6. Kang J.S. Practical and provable security against differential and linear cryptanalysis for substitution-permutation networks / J.S. Kang, S. Hong, S. Lee, O. Yi, C. Park, J. Lim // ETRI Journal. — 2001. — no. 23. — P. 158–167.

7. Hong S. Provable security against differential and linear cryptanalysis for the spn structure / Hong S // Fast Software Encryption. – FSE'00 Proceedings. — 2001. — P. 273–283.

8. Biham E. Differential cryptanalysis of des-like cryptosystems /Biham E., A. Shamir //Journal of Cryptology. — 1991. — Vol. 4, no. 1. — P. 3–72.

9. Biham E. Differential cryptanalysis of the full 16-round des /Biham E., A. Shamir //Advances in Cryptology – CRYPTO'92. — 1993. — P. 487–496.

10. Kanda M. Practical security evaluation against differential and linear cryptanalyses for feistel ciphers with spn round function /Kanda M. //Selected Areas in Cryptography. – SAC 2000, Proceedings. — 2001. — P. 324–338.

11. Ковальчук Л.В. Обобщенные марковские шифры: построение оценки практической стойкости относительно дифференциального криптоанализа / Ковальчук Л.В.//Математика и безопасность информационных технологий. Материалы конференции в МГУ 25 – 27 октября 2006. — 2007. — С. 595–599.

12. Ковальчук Л.В. О криптографических свойствах нового нацинального стандарта шифрования украины /Ковальчук Л.В. //Кибернетика и системный анализ. — 2016. — Т. 52, № 3. — С. 16–32.

13. Яковлєв С.В. Уточнені оцінки стійкості алгоритму шифрування ДСТУ 7624:2014 до диференціального та лінійного криптоаналізу /Яковлєв С.В. //Інформаційні технології та комп'ютерна інженерія. — 2015. — С. 176–178.

14. Яковлєв С.В. Аналітичні оцінки стійкості немарковських симетричних блочних шифрів до диференціального криптоаналізу : кандидатська дисертація / Яковлєв С.В. — К.: НТУУ «КПІ», 2016. — 160 с.

15. НДР «Дослідження та застосування сучасних математичних методів окремих перетворень у системах криптографічного захисту інформації» (шифр «Мокрель»), держ. реестр №0115-004118. — К.: НТУУ «КПІ», 2016. — 103 с..

16. Knudsen L.R. Practically secure feistel cipher / Knudsen L.R. //Software Encryption. – FSE'94, Proceedings. — 1994. — P. 211–221.

Додаток А

Лістинг програми

Лістинг файлу Kalyna.cpp — реалізація алг. 2.1, алг. 2.2, алг 2.3, предобчислення W.

```
#include <iostream>
#include <vector>
#include <math.h>
#include <stdint.h>

#include <stdlib.h>
#include <string.h>

#include <stdio.h>
#include <memory.h>
#include <limits.h>
#include <algorithm>
#include <omp.h>
#include <fstream>

#include <map>
//#include "BigIntegerLibrary.hh"
#include <iomanip>



using namespace std;



uint8_t sboxes_enc[4][256] = {
        {
                0xa8, 0x43, 0x5f, 0x06, 0x6b, 0x75, 0x6c, 0x59, 0x71, 0xdf, 0x87, 0x95, 0x17, 0xf0, 0xd8, 0x09,
                0x6d, 0xf3, 0x1d, 0xcb, 0xc9, 0x4d, 0x2c, 0xaf, 0x79, 0xe0, 0x97, 0xfd, 0x6f, 0x4b, 0x45, 0x39,
        0x3e, 0xdd, 0xa3, 0x4f, 0xb4, 0xb6, 0x9a, 0x0e, 0x1f, 0xbf, 0x15, 0xe1, 0x49, 0xd2, 0x93, 0xc6,
        0x92, 0x72, 0x9e, 0x61, 0xd1, 0x63, 0xfa, 0xee, 0xf4, 0x19, 0xd5, 0xad, 0x58, 0xa4, 0xbb, 0xa1,
        0xdc, 0xf2, 0x83, 0x37, 0x42, 0xe4, 0x7a, 0x32, 0x9c, 0xcc, 0xab, 0x4a, 0x8f, 0x6e, 0x04, 0x27,
        0x2e, 0xe7, 0xe2, 0x5a, 0x96, 0x16, 0x23, 0x2b, 0xc2, 0x65, 0x66, 0x0f, 0xbc, 0xa9, 0x47, 0x41,
        0x34, 0x48, 0xfc, 0xb7, 0x6a, 0x88, 0xa5, 0x53, 0x86, 0xf9, 0x5b, 0xdb, 0x38, 0x7b, 0xc3, 0x1e,
        0x22, 0x33, 0x24, 0x28, 0x36, 0xc7, 0xb2, 0x3b, 0x8e, 0x77, 0xba, 0xf5, 0x14, 0x9f, 0x08, 0x55,
        0x9b, 0x4c, 0xfe, 0x60, 0x5c, 0xda, 0x18, 0x46, 0xcd, 0x7d, 0x21, 0xb0, 0x3f, 0x1b, 0x89, 0xff,
        0xeb, 0x84, 0x69, 0x3a, 0x9d, 0xd7, 0xd3, 0x70, 0x67, 0x40, 0xb5, 0xde, 0x5d, 0x30, 0x91, 0xb1,
        0x78, 0x11, 0x01, 0xe5, 0x00, 0x68, 0x98, 0xa0, 0xc5, 0x02, 0xa6, 0x74, 0x2d, 0x0b, 0xa2, 0x76,
        0xb3, 0xbe, 0xce, 0xbd, 0xae, 0xe9, 0x8a, 0x31, 0x1c, 0xec, 0xf1, 0x99, 0x94, 0xaa, 0xf6, 0x26,
        0x2f, 0xef, 0xe8, 0x8c, 0x35, 0x03, 0xd4, 0x7f, 0xfb, 0x05, 0xc1, 0x5e, 0x90, 0x20, 0x3d, 0x82,
        0xf7, 0xea, 0x0a, 0x0d, 0x7e, 0xf8, 0x50, 0x1a, 0xc4, 0x07, 0x57, 0xb8, 0x3c, 0x62, 0xe3, 0xc8,
        0xac, 0x52, 0x64, 0x10, 0xd0, 0xd9, 0x13, 0x0c, 0x12, 0x29, 0x51, 0xb9, 0xcf, 0xd6, 0x73, 0x8d,
        0x81, 0x54, 0xc0, 0xed, 0x4e, 0x44, 0xa7, 0x2a, 0x85, 0x25, 0xe6, 0xca, 0x7c, 0x8b, 0x56, 0x80
        },
        {
                0xce, 0xbb, 0xeb, 0x92, 0xea, 0xcb, 0x13, 0xc1, 0xe9, 0x3a, 0xd6, 0xb2, 0xd2, 0x90, 0x17, 0xf8,
                0x42, 0x15, 0x56, 0xb4, 0x65, 0x1c, 0x88, 0x43, 0xc5, 0x5c, 0x36, 0xba, 0xf5, 0x57, 0x67, 0x8d,
        0x31, 0xf6, 0x64, 0x58, 0x9e, 0xf4, 0x22, 0xaa, 0x75, 0x0f, 0x02, 0xb1, 0xdf, 0x6d, 0x73, 0x4d,
        0x7c, 0x26, 0x2e, 0xf7, 0x08, 0x5d, 0x44, 0x3e, 0x9f, 0x14, 0xc8, 0xae, 0x54, 0x10, 0xd8, 0xbc,
        0x1a, 0x6b, 0x69, 0xf3, 0xbd, 0x33, 0xab, 0xfa, 0xd1, 0x9b, 0x68, 0x4e, 0x16, 0x95, 0x91, 0xee,
        0x4c, 0x63, 0x8e, 0x5b, 0xcc, 0x3c, 0x19, 0xa1, 0x81, 0x49, 0x7b, 0xd9, 0x6f, 0x37, 0x60, 0xca,
        0xe7, 0x2b, 0x48, 0xfd, 0x96, 0x45, 0xfc, 0x41, 0x12, 0x0d, 0x79, 0xe5, 0x89, 0x8c, 0xe3, 0x20,
        0x30, 0xdc, 0xb7, 0x6c, 0x4a, 0xb5, 0x3f, 0x97, 0xd4, 0x62, 0x2d, 0x06, 0xa4, 0xa5, 0x83, 0x5f,
        0x2a, 0xda, 0xc9, 0x00, 0x7e, 0xa2, 0x55, 0xbf, 0x11, 0xd5, 0x9c, 0xcf, 0x0e, 0x0a, 0x3d, 0x51,
        0x7d, 0x93, 0x1b, 0xfe, 0xc4, 0x47, 0x09, 0x86, 0x0b, 0x8f, 0x9d, 0x6a, 0x07, 0xb9, 0xb0, 0x98,
        0x18, 0x32, 0x71, 0x4b, 0xef, 0x3b, 0x70, 0xa0, 0xe4, 0x40, 0xff, 0xc3, 0xa9, 0xe6, 0x78, 0xf9,
        0x8b, 0x46, 0x80, 0x1e, 0x38, 0xe1, 0xb8, 0xa8, 0xe0, 0x0c, 0x23, 0x76, 0x1d, 0x25, 0x24, 0x05,
        0xf1, 0x6e, 0x94, 0x28, 0x9a, 0x84, 0xe8, 0xa3, 0x4f, 0x77, 0xd3, 0x85, 0xe2, 0x52, 0xf2, 0x82,
        0x50, 0x7a, 0x2f, 0x74, 0x53, 0xb3, 0x61, 0xaf, 0x39, 0x35, 0xde, 0xcd, 0x1f, 0x99, 0xac, 0xad,
        0x72, 0x2c, 0xdd, 0xd0, 0x87, 0xbe, 0x5e, 0xa6, 0xec, 0x04, 0xc6, 0x03, 0x34, 0xfb, 0xdb, 0x59,
        0xb6, 0xc2, 0x01, 0xf0, 0x5a, 0xed, 0xa7, 0x66, 0x21, 0x7f, 0x8a, 0x27, 0xc7, 0xc0, 0x29, 0xd7
        },
        {
                0x93, 0xd9, 0x9a, 0xb5, 0x98, 0x22, 0x45, 0xfc, 0xba, 0x6a, 0xdf, 0x02, 0x9f, 0xdc, 0x51, 0x59,
                0x4a, 0x17, 0x2b, 0xc2, 0x94, 0xf4, 0xbb, 0xa3, 0x62, 0xe4, 0x71, 0xd4, 0xcd, 0x70, 0x16, 0xe1,
        0x49, 0x3c, 0xc0, 0xd8, 0x5c, 0x9b, 0xad, 0x85, 0x53, 0xa1, 0x7a, 0xc8, 0x2d, 0xe0, 0xd1, 0x72,
        0xa6, 0x2c, 0xc4, 0xe3, 0x76, 0x78, 0xb7, 0xb4, 0x09, 0x3b, 0x0e, 0x41, 0x4c, 0xde, 0xb2, 0x90,
        0x25, 0xa5, 0xd7, 0x03, 0x11, 0x00, 0xc3, 0x2e, 0x92, 0xef, 0x4e, 0x12, 0x9d, 0x7d, 0xcb, 0x35,
        0x10, 0xd5, 0x4f, 0x9e, 0x4d, 0xa9, 0x55, 0xc6, 0xd0, 0x7b, 0x18, 0x97, 0xd3, 0x36, 0xe6, 0x48,
        0x56, 0x81, 0x8f, 0x77, 0xcc, 0x9c, 0xb9, 0xe2, 0xac, 0xb8, 0x2f, 0x15, 0xa4, 0x7c, 0xda, 0x38,
        0x1e, 0x0b, 0x05, 0xd6, 0x14, 0x6e, 0x6c, 0x7e, 0x66, 0xfd, 0xb1, 0xe5, 0x60, 0xaf, 0x5e, 0x33,
        0x87, 0xc9, 0xf0, 0x5d, 0x6d, 0x3f, 0x88, 0x8d, 0xc7, 0xf7, 0x1d, 0xe9, 0xec, 0xed, 0x80, 0x29,
        0x27, 0xcf, 0x99, 0xa8, 0x50, 0x0f, 0x37, 0x24, 0x28, 0x30, 0x95, 0xd2, 0x3e, 0x5b, 0x40, 0x83,
        0xb3, 0x69, 0x57, 0x1f, 0x07, 0x1c, 0x8a, 0xbc, 0x20, 0xeb, 0xce, 0x8e, 0xab, 0xee, 0x31, 0xa2,
        0x73, 0xf9, 0xca, 0x3a, 0x1a, 0xfb, 0x0d, 0xc1, 0xfe, 0xfa, 0xf2, 0x6f, 0xbd, 0x96, 0xdd, 0x43,
        0x52, 0xb6, 0x08, 0xf3, 0xae, 0xbe, 0x19, 0x89, 0x32, 0x26, 0xb0, 0xea, 0x4b, 0x64, 0x84, 0x82,
        0x6b, 0xf5, 0x79, 0xbf, 0x01, 0x5f, 0x75, 0x63, 0x1b, 0x23, 0x3d, 0x68, 0x2a, 0x65, 0xe8, 0x91,
        0xf6, 0xff, 0x13, 0x58, 0xf1, 0x47, 0x0a, 0x7f, 0xc5, 0xa7, 0xe7, 0x61, 0x5a, 0x06, 0x46, 0x44,
```

```
        0x42, 0x04, 0xa0, 0xdb, 0x39, 0x86, 0x54, 0xaa, 0x8c, 0x34, 0x21, 0x8b, 0xf8, 0x0c, 0x74, 0x67
        },
        {
                0x68, 0x8d, 0xca, 0x4d, 0x73, 0x4b, 0x4e, 0x2a, 0xd4, 0x52, 0x26, 0xb3, 0x54, 0x1e, 0x19, 0x1f,
                0x22, 0x03, 0x46, 0x3d, 0x2d, 0x4a, 0x53, 0x83, 0x13, 0x8a, 0xb7, 0xd5, 0x25, 0x79, 0xf5, 0xbd,
        0x58, 0x2f, 0x0d, 0x02, 0xed, 0x51, 0x9e, 0x11, 0xf2, 0x3e, 0x55, 0x5e, 0xd1, 0x16, 0x3c, 0x66,
        0x70, 0x5d, 0xf3, 0x45, 0x40, 0xcc, 0xe8, 0x94, 0x56, 0x08, 0xce, 0x1a, 0x3a, 0xd2, 0xe1, 0xdf,
        0xb5, 0x38, 0x6e, 0x0e, 0xe5, 0xf4, 0xf9, 0x86, 0xe9, 0x4f, 0xd6, 0x85, 0x23, 0xcf, 0x32, 0x99,
        0x31, 0x14, 0xae, 0xee, 0xc8, 0x48, 0xd3, 0x30, 0xa1, 0x92, 0x41, 0xb1, 0x18, 0xc4, 0x2c, 0x71,
        0x72, 0x44, 0x15, 0xfd, 0x37, 0xbe, 0x5f, 0xaa, 0x9b, 0x88, 0xd8, 0xab, 0x89, 0x9c, 0xfa, 0x60,
        0xea, 0xbc, 0x62, 0x0c, 0x24, 0xa6, 0xa8, 0xec, 0x67, 0x20, 0xdb, 0x7c, 0x28, 0xdd, 0xac, 0x5b,
        0x34, 0x7e, 0x10, 0xf1, 0x7b, 0x8f, 0x63, 0xa0, 0x05, 0x9a, 0x43, 0x77, 0x21, 0xbf, 0x27, 0x09,
        0xc3, 0x9f, 0xb6, 0xd7, 0x29, 0xc2, 0xeb, 0xc0, 0xa4, 0x8b, 0x8c, 0x1d, 0xfb, 0xff, 0xc1, 0xb2,
        0x97, 0x2e, 0xf8, 0x65, 0xf6, 0x75, 0x07, 0x04, 0x49, 0x33, 0xe4, 0xd9, 0xb9, 0xd0, 0x42, 0xc7,
        0x6c, 0x90, 0x00, 0x8e, 0x6f, 0x50, 0x01, 0xc5, 0xda, 0x47, 0x3f, 0xcd, 0x69, 0xa2, 0xe2, 0x7a,
        0xa7, 0xc6, 0x93, 0x0f, 0x0a, 0x06, 0xe6, 0x2b, 0x96, 0xa3, 0x1c, 0xaf, 0x6a, 0x12, 0x84, 0x39,
        0xe7, 0xb0, 0x82, 0xf7, 0xfe, 0x9d, 0x87, 0x5c, 0x81, 0x35, 0xde, 0xb4, 0xa5, 0xfc, 0x80, 0xef,
        0xcb, 0xbb, 0x6b, 0x76, 0xba, 0x5a, 0x7d, 0x78, 0x0b, 0x95, 0xe3, 0xad, 0x74, 0x98, 0x3b, 0x36,
        0x64, 0x6d, 0xdc, 0xf0, 0x59, 0xa9, 0x4c, 0x17, 0x7f, 0x91, 0xb8, 0xc9, 0x57, 0x1b, 0xe0, 0x61
        }
};


uint8_t sboxes_dec[4][256] = {
        {
                0xa4, 0xa2, 0xa9, 0xc5, 0x4e, 0xc9, 0x03, 0xd9, 0x7e, 0x0f, 0xd2, 0xad, 0xe7, 0xd3, 0x27, 0x5b,
                0xe3, 0xa1, 0xe8, 0xe6, 0x7c, 0x2a, 0x55, 0x0c, 0x86, 0x39, 0xd7, 0x8d, 0xb8, 0x12, 0x6f, 0x28,
        0xcd, 0x8a, 0x70, 0x56, 0x72, 0xf9, 0xbf, 0x4f, 0x73, 0xe9, 0xf7, 0x57, 0x16, 0xac, 0x50, 0xc0,
        0x9d, 0xb7, 0x47, 0x71, 0x60, 0xc4, 0x74, 0x43, 0x6c, 0x1f, 0x93, 0x77, 0xdc, 0xce, 0x20, 0x8c,
        0x99, 0x5f, 0x44, 0x01, 0xf5, 0x1e, 0x87, 0x5e, 0x61, 0x2c, 0x4b, 0x1d, 0x81, 0x15, 0xf4, 0x23,
        0xd6, 0xea, 0xe1, 0x67, 0xf1, 0x7f, 0xfe, 0xda, 0x3c, 0x07, 0x53, 0x6a, 0x84, 0x9c, 0xcb, 0x02,
        0x83, 0x33, 0xdd, 0x35, 0xe2, 0x59, 0x5a, 0x98, 0xa5, 0x92, 0x64, 0x04, 0x06, 0x10, 0x4d, 0x1c,
        0x97, 0x08, 0x31, 0xee, 0xab, 0x05, 0xaf, 0x79, 0xa0, 0x18, 0x46, 0x6d, 0xfc, 0x89, 0xd4, 0xc7,
        0xff, 0xf0, 0xcf, 0x42, 0x91, 0xf8, 0x68, 0x0a, 0x65, 0x8e, 0xb6, 0xfd, 0xc3, 0xef, 0x78, 0x4c,
        0xcc, 0x9e, 0x30, 0x2e, 0xbc, 0x0b, 0x54, 0x1a, 0xa6, 0xbb, 0x26, 0x80, 0x48, 0x94, 0x32, 0x7d,
        0xa7, 0x3f, 0xae, 0x22, 0x3d, 0x66, 0xaa, 0xf6, 0x00, 0x5d, 0xbd, 0x4a, 0xe0, 0x3b, 0xb4, 0x17,
        0x8b, 0x9f, 0x76, 0xb0, 0x24, 0x9a, 0x25, 0x63, 0xdb, 0xeb, 0x7a, 0x3e, 0x5c, 0xb3, 0xb1, 0x29,
        0xf2, 0xca, 0x58, 0x6e, 0xd8, 0xa8, 0x2f, 0x75, 0xdf, 0x14, 0xfb, 0x13, 0x49, 0x88, 0xb2, 0xec,
        0xe4, 0x34, 0x2d, 0x96, 0xc6, 0x3a, 0xed, 0x95, 0x0e, 0xe5, 0x85, 0x6b, 0x40, 0x21, 0x9b, 0x09,
        0x19, 0x2b, 0x52, 0xde, 0x45, 0xa3, 0xfa, 0x51, 0xc2, 0xb5, 0xd1, 0x90, 0xb9, 0xf3, 0x37, 0xc1,
        0x0d, 0xba, 0x41, 0x11, 0x38, 0x7b, 0xbe, 0xd0, 0xd5, 0x69, 0x36, 0xc8, 0x62, 0x1b, 0x82, 0x8f
        },
        {
                0x83, 0xf2, 0x2a, 0xeb, 0xe9, 0xbf, 0x7b, 0x9c, 0x34, 0x96, 0x8d, 0x98, 0xb9, 0x69, 0x8c, 0x29,
                0x3d, 0x88, 0x68, 0x06, 0x39, 0x11, 0x4c, 0x0e, 0xa0, 0x56, 0x40, 0x92, 0x15, 0xbc, 0xb3, 0xdc,
        0x6f, 0xf8, 0x26, 0xba, 0xbe, 0xbd, 0x31, 0xfb, 0xc3, 0xfe, 0x80, 0x61, 0xe1, 0x7a, 0x32, 0xd2,
        0x70, 0x20, 0xa1, 0x45, 0xec, 0xd9, 0x1a, 0x5d, 0xb4, 0xd8, 0x09, 0xa5, 0x55, 0x8e, 0x37, 0x76,
        0xa9, 0x67, 0x10, 0x17, 0x36, 0x65, 0xb1, 0x95, 0x62, 0x59, 0x74, 0xa3, 0x50, 0x2f, 0x4b, 0xc8,
        0xd0, 0x8f, 0xcd, 0xd4, 0x3c, 0x86, 0x12, 0x1d, 0x23, 0xef, 0xf4, 0x53, 0x19, 0x35, 0xe6, 0x7f,
        0x5e, 0xd6, 0x79, 0x51, 0x22, 0x14, 0xf7, 0x1e, 0x4a, 0x42, 0x9b, 0x41, 0x73, 0x2d, 0xc1, 0x5c,
        0xa6, 0xa2, 0xe0, 0x2e, 0xd3, 0x28, 0xbb, 0xc9, 0xae, 0x6a, 0xd1, 0x5a, 0x30, 0x90, 0x84, 0xf9,
        0xb2, 0x58, 0xcf, 0x7e, 0xc5, 0xcb, 0x97, 0xe4, 0x16, 0x6c, 0xfa, 0xb0, 0x6d, 0x1f, 0x52, 0x99,
        0x0d, 0x4e, 0x03, 0x91, 0xc2, 0x4d, 0x64, 0x77, 0x9f, 0xdd, 0xc4, 0x49, 0x8a, 0x9a, 0x24, 0x38,
        0xa7, 0x57, 0x85, 0xc7, 0x7c, 0x7d, 0xe7, 0xf6, 0xb7, 0xac, 0x27, 0x46, 0xde, 0xdf, 0x3b, 0xd7,
        0x9e, 0x2b, 0x0b, 0xd5, 0x13, 0x75, 0xf0, 0x72, 0xb6, 0x9d, 0x1b, 0x01, 0x3f, 0x44, 0xe5, 0x87,
        0xfd, 0x07, 0xf1, 0xab, 0x94, 0x18, 0xea, 0xfc, 0x3a, 0x82, 0x5f, 0x05, 0x54, 0xdb, 0x00, 0x8b,
        0xe3, 0x48, 0x0c, 0xca, 0x78, 0x89, 0x0a, 0xff, 0x3e, 0x5b, 0x81, 0xee, 0x71, 0xe2, 0xda, 0x2c,
        0xb8, 0xb5, 0xcc, 0x6e, 0xa8, 0x6b, 0xad, 0x60, 0xc6, 0x08, 0x04, 0x02, 0xe8, 0xf5, 0x4f, 0xa4,
        0xf3, 0xc0, 0xce, 0x43, 0x25, 0x1c, 0x21, 0x33, 0x0f, 0xaf, 0x47, 0xed, 0x66, 0x63, 0x93, 0xaa
        },
        {
                0x45, 0xd4, 0x0b, 0x43, 0xf1, 0x72, 0xed, 0xa4, 0xc2, 0x38, 0xe6, 0x71, 0xfd, 0xb6, 0x3a, 0x95,
                0x50, 0x44, 0x4b, 0xe2, 0x74, 0x6b, 0x1e, 0x11, 0x5a, 0xc6, 0xb4, 0xd8, 0xa5, 0x8a, 0x70, 0xa3,
        0xa8, 0xfa, 0x05, 0xd9, 0x97, 0x40, 0xc9, 0x90, 0x98, 0x8f, 0xdc, 0x12, 0x31, 0x2c, 0x47, 0x6a,
        0x99, 0xae, 0xc8, 0x7f, 0xf9, 0x4f, 0x5d, 0x96, 0x6f, 0xf4, 0xb3, 0x39, 0x21, 0xda, 0x9c, 0x85,
        0x9e, 0x3b, 0xf0, 0xbf, 0xef, 0x06, 0xee, 0xe5, 0x5f, 0x20, 0x10, 0xcc, 0x3c, 0x54, 0x4a, 0x52,
        0x94, 0x0e, 0xc0, 0x28, 0xf6, 0x56, 0x60, 0xa2, 0xe3, 0x0f, 0xec, 0x9d, 0x24, 0x83, 0x7e, 0xd5,
        0x7c, 0xeb, 0x18, 0xd7, 0xcd, 0xdd, 0x78, 0xff, 0xdb, 0xa1, 0x09, 0xd0, 0x76, 0x84, 0x75, 0xbb,
        0x1d, 0x1a, 0x2f, 0xb0, 0xfe, 0xd6, 0x34, 0x63, 0x35, 0xd2, 0x2a, 0x59, 0x6d, 0x4d, 0x77, 0xe7,
        0x8e, 0x61, 0xcf, 0x9f, 0xce, 0x27, 0xf5, 0x80, 0x86, 0xc7, 0xa6, 0xfb, 0xf8, 0x87, 0xab, 0x62,
        0x3f, 0xdf, 0x48, 0x00, 0x14, 0x9a, 0xbd, 0x5b, 0x04, 0x92, 0x02, 0x25, 0x65, 0x4c, 0x53, 0x0c,
        0xf2, 0x29, 0xaf, 0x17, 0x6c, 0x41, 0x30, 0xe9, 0x93, 0x55, 0xf7, 0xac, 0x68, 0x26, 0xc4, 0x7d,
        0xca, 0x7a, 0x3e, 0xa0, 0x37, 0x03, 0xc1, 0x36, 0x69, 0x66, 0x08, 0x16, 0xa7, 0xbc, 0xc5, 0xd3,
        0x22, 0xb7, 0x13, 0x46, 0x32, 0xe8, 0x57, 0x88, 0x2b, 0x81, 0xb2, 0x4e, 0x64, 0x1c, 0xaa, 0x91,
        0x58, 0x2e, 0x9b, 0x5c, 0x1b, 0x51, 0x73, 0x42, 0x23, 0x01, 0x6e, 0xf3, 0x0d, 0xbe, 0x3d, 0x0a,
        0x2d, 0x1f, 0x67, 0x33, 0x19, 0x7b, 0x5e, 0xea, 0xde, 0x8b, 0xcb, 0xa9, 0x8c, 0x8d, 0xad, 0x49,
        0x82, 0xe4, 0xba, 0xc3, 0x15, 0xd1, 0xe0, 0x89, 0xfc, 0xb1, 0xb9, 0xb5, 0x07, 0x79, 0xb8, 0xe1
        },
        {
                0xb2, 0xb6, 0x23, 0x11, 0xa7, 0x88, 0xc5, 0xa6, 0x39, 0x8f, 0xc4, 0xe8, 0x73, 0x22, 0x43, 0xc3,
                0x82, 0x27, 0xcd, 0x18, 0x51, 0x62, 0x2d, 0xf7, 0x5c, 0x0e, 0x3b, 0xfd, 0xca, 0x9b, 0x0d, 0x0f,
        0x79, 0x8c, 0x10, 0x4c, 0x74, 0x1c, 0x0a, 0x8e, 0x7c, 0x94, 0x07, 0xc7, 0x5e, 0x14, 0xa1, 0x21,
        0x57, 0x50, 0x4e, 0xa9, 0x80, 0xd9, 0xef, 0x64, 0x41, 0xcf, 0x3c, 0xee, 0x2e, 0x13, 0x29, 0xba,
        0x34, 0x5a, 0xae, 0x8a, 0x61, 0x33, 0x12, 0xb9, 0x55, 0xa8, 0x15, 0x05, 0xf6, 0x03, 0x06, 0x49,
        0xb5, 0x25, 0x09, 0x16, 0x0c, 0x2a, 0x38, 0xfc, 0x20, 0xf4, 0xe5, 0x7f, 0xd7, 0x31, 0x2b, 0x66,
        0x6f, 0xff, 0x72, 0x86, 0xf0, 0xa3, 0x2f, 0x78, 0x00, 0xbc, 0xcc, 0xe2, 0xb0, 0xf1, 0x42, 0xb4,
        0x30, 0x5f, 0x60, 0x04, 0xec, 0xa5, 0xe3, 0x8b, 0xe7, 0x1d, 0xbf, 0x84, 0x7b, 0xe6, 0x81, 0xf8,
        0xde, 0xd8, 0xd2, 0x17, 0xce, 0x4b, 0x47, 0xd6, 0x69, 0x6c, 0x19, 0x99, 0x9a, 0x01, 0xb3, 0x85,
        0xb1, 0xf9, 0x59, 0xc2, 0x37, 0xe9, 0xc8, 0xa0, 0xed, 0x4f, 0x89, 0x68, 0x6d, 0xd5, 0x26, 0x91,
        0x87, 0x58, 0xbd, 0xc9, 0x98, 0xdc, 0x75, 0xc0, 0x76, 0xf5, 0x67, 0x6b, 0x7e, 0xeb, 0x52, 0xcb,
        0xd1, 0x5b, 0x9f, 0x0b, 0xdb, 0x40, 0x92, 0x1a, 0xfa, 0xac, 0xe4, 0xe1, 0x71, 0x1f, 0x65, 0x8d,
        0x97, 0x9e, 0x95, 0x90, 0x5d, 0xb7, 0xc1, 0xaf, 0x54, 0xfb, 0x02, 0xe0, 0x35, 0xbb, 0x3a, 0x4d,
        0xad, 0x2c, 0x3d, 0x56, 0x08, 0x1b, 0x4a, 0x93, 0x6a, 0xab, 0xb8, 0x7a, 0xf2, 0x7d, 0xda, 0x3f,
        0xfe, 0x3e, 0xbe, 0xea, 0xaa, 0x44, 0xc6, 0xd0, 0x36, 0x48, 0x70, 0x96, 0x77, 0x24, 0x53, 0xdf,
        0xf3, 0x83, 0x28, 0x32, 0x45, 0x1e, 0xa4, 0xd3, 0xa2, 0x46, 0x6e, 0x9c, 0xdd, 0x63, 0xd4, 0x9d
        }
};


long double ds0[256][256][4];
```

```
struct  MyPair
{
        long double p;
        uint64_t N;
};


vector<MyPair> dpN[4];
vector<MyPair> U[17];
vector<MyPair> U256[257];
//BigInteger W[256][256];
uint64_t Wint64[256][256];
int big_w[65536];
vector<uint16_t> W_k_b[65536];
float W_log[256][256];


int min(int a,int b)
{
        if (a < b) return a;
        else return b;
}



uint64_t get_Wp(int a, int b, int m = 8)
{
        uint64_t two = 2;
        uint64_t result = 1;
        if (weight[a] > (m - weight[b]))
        {
                for (int i = 0;i < 8 * (weight[a] + weight[b] - m);i++)
                {
                        result *= two;
                }
                return result;
        }
        else return 1;
}




uint64_t get_sumW(int a, int b)
{


        int wtb = weight[b];
        int m = 4;
        //int u = 8;


        uint64_t result = 0;

        for (int t = 0; t < wtb + 1; t++)
        {
                for (int c = 0; c < 16;c++)
                {
                        if ((domination[c][b] == 1) && (weight[c] == t))
                        {

                                if (!((wtb - t) % 2))
                                {
                                        result += get_Wp(a, c, m);
                                }
                                else
                                {
                                        result -= get_Wp(a, c, m);
                                }

                        }
                }
        }
        return result;
}

uint64_t get_sumWn(int a, int b)
{


        int wta = weight[a];
        //int m = 8;
        //int u = 8;


        uint64_t result = 0;

        for (int t = 0; t < wta + 1; t++)
        {
                for (int c = 0; c < 16;c++)
                {
                        if ((domination[c][a]) && (weight[c] == t))
                        {
                                //long long pow = weight[b] + weight[c] - m;
                                if (!((wta - t) % 2))
                                {
                                        result += get_sumW(c, b);
```

```
                                }
                                else
                                {
                                        result -= get_sumW(c, b);
                                }

                        }
                }
        }
        return result;
}


long double ds(uint8_t a, uint8_t b,uint8_t i=0)
{
        uint64_t count = 0;
        if ((a==0) && (b==0)) return 1;
        if ((a==0) || (b==0)) return 0;
        //if ((a || b) == 0) return 0;
        for (int x = 0; x < 256; x++)
        {
                if ((sboxes_enc[i][x ^ a]) == (sboxes_enc[i][x] ^ b)) count++;
        }
        long double result = count / (1.0 * 256);
        return result;
}

int compare(const void * a, const void * b)
{
        return (*(int*)b - *(int*)a);
}




vector<MyPair> compose(vector<MyPair> a, vector<MyPair> b)
{

        vector<MyPair> tmp,result;
        MyPair t;
        for (unsigned int i = 0;i < a.size();i++)
                for (unsigned int j = 0;j < b.size();j++)
                {
                        t.p = a[i].p*b[j].p;
                        t.N = a[i].N*b[j].N;
                        tmp.push_back(t);
                }

        for (unsigned int i = 0;i < tmp.size();i++)
                for (unsigned int j = 0;j < tmp.size();j++)
                {
                        if (tmp[i].p > tmp[j].p)
                        {
                                t = tmp[j];
                                tmp[j] = tmp[i];
                                tmp[i] = t;
                        }
                }
        t = tmp[0];
        for (unsigned int i = 1;i < tmp.size();i++)
        {
                if (tmp[i].p == t.p)
                {
                        t.N += tmp[i].N;
                }
                else
                {
                        result.push_back(t);
                        t = tmp[i];
                }
        }
        result.push_back(t);
        return result;
}

vector<MyPair> u(int a)
{
        vector<int> pos;
        int i = 0;
        //cout<<"F"<<a<<endl;
        vector<MyPair> result;
        while (a)
        {
                if(a % 2) pos.push_back(i);
                a = a / 2;
                i++;
        }
        result = dpN[pos[0]];

        for (unsigned int i = 1;i < pos.size();i++)
        {
                result = compose(result, dpN[pos[i]]);
        }

        return result;
}

int get_weight(uint64_t t)
{
```

```
        int result = 0;
        while (t)
        {
                result += t % 2;
                t >>= 1;
        }
        return result;
}



long double m[256];
long double ub[256][256];

void UB_42_2(long double QD)
{

        for (int i = 0;i < 256;i++)
        {
                m[i] = 0;
        }
        long double result[256];

        for(int i=0;i<256;i++)
        {
                ub[i][0]=0;
                ub[0][i]=0;
        }

        ub[0][0]=1;

        for(int a=1;a<256;a++)
        {

                for(int b=1;b<256;b++)
                {

                        uint8_t a1_t, a2_t, a3_t, a4_t, b1, b2, a1, a2;
                                a1_t = a % 4;
                                a2_t = (a >> 2) % 4;
                                a3_t = (a >> 4) % 4;
                                a4_t = (a >> 6) % 4;
                                b1 = b % 16;
                                b2 = (b >> 4) % 16;
                                a1 = a1_t + (a4_t << 2);
                                a2 = a3_t + (a2_t << 2);


                        uint64_t  t2 = 1;
                        t2 <<= 40;



                        uint64_t big_wIb = (1 << weight[b]);
                        uint64_t big_wIa = (1 << weight[a]);
                        long double t1a = (1 / (1.0 * big_wIa));
                        t1a /= big_wIa;
                        t1a *= t1a;
                        t1a /= big_wIa;
                        long double t1b = (1 / (1.0 * big_wIb));
                        t1b /= big_wIb;
                        t1b *= t1b;
                        t1b /= big_wIb;
                        uint64_t m_c, i = 0, j = 0;
                        uint64_t W_full=Wint64[a1][b1]*Wint64[a2][b2];
                        long double sum=0;
                        vector<MyPair> u_a=U256[a];
                        vector<MyPair> u_b=U256[b];
                        result[b]=0;
                        //double result=0;
                                if (W_full != 0)
                                {
                                        m_c = min(u_a[i].N, u_b[j].N);
                                        W_full-= m_c;
                                        while (m_c < W_full)
                                        {
                                                        sum += u_a[i].p*u_b[j].p*m_c;
                                                        if (u_a[i].N > m_c)
                                                        {
                                                                u_a[i].N -= m_c;
                                                                if (u_b.size() > j + 1) {
                                                                        j++;
                                                                }
                                                                else
                                                                {
                                                                        break;
                                                                }
                                                        }
                                                        else
                                                        {
                                                                u_b[j].N -= m_c;
                                                                if (u_a.size() > i + 1) {
                                                                        i++;
                                                                }
                                                                else
                                                                {
                                                                        break;
                                                                }
                                                        }
                                                        m_c = min(u_a[i].N, u_b[j].N);
```

```
                                                W_full-= m_c;
                                        }
                                        //sum+= u_a[i].p*u_b[j].p*W_full;
                                }

                                result[b]=min(QD,min(t1a,t1b));
                                //cout<<result[b]<<endl;
                                //      cout<<sum<<endl;
                                result[b]=min(sum,result[b]);
                                ub[a][b]=result[b];
                                m[b] = max(m[b], result[b]);
                        }
                }

}


void UB4_2_3()
{
        long double result[256],next_ub[256][256];
        for(int i=0;i<256;i++)
                {
                        m[i]=0;
                }
        for(int i=1;i<256;i++)
                for(int j=1;j<256;j++)
                {
                        m[i]=max(ub[i][j],m[i]);
                }



        for(int a=1;a<256;a++)
        {
                //long double t=0;
                for(int b=1;b<256;b++)
                {

                        long double temp=0;
                        uint32_t big_wIb = (1 << weight[b]);
                        long double t1b = (1 / (1.0 * big_wIb));
                                        t1b /= big_wIb;
                                        t1b *= t1b;
                                        t1b /= big_wIb;

                                        uint64_t big_wIa = (1 << weight[a]);
                                        long double t1a = (1 / (1.0 * big_wIa));
                                        t1a /= big_wIa;
                                        t1a *= t1a;
                                        t1a /= big_wIa;



                        for(unsigned int k=1;k<256;k++)
                        {
                                uint8_t k1_t, k2_t, k3_t, k4_t, b1, b2, k1, k2;
                                k1_t = k % 4;
                                k2_t = (k >> 2) % 4;
                                k3_t = (k >> 4) % 4;
                                k4_t = (k >> 6) % 4;
                                b1 = b % 16;
                                b2 = (b >> 4) % 16;
                                k1 = k1_t + (k4_t << 2);
                                k2 = k3_t + (k2_t << 2);




                                long double part_sum=0;
                                if((Wint64[k1][b1]>0) && (Wint64[k2][b2]>0))
                                {
                                        uint64_t W_n=Wint64[k1][b1]*Wint64[k2][b2];
                                                vector<MyPair> u_b=U256[b];
                                                int i=0;
                                                while((W_n>u_b[i].N) && (i+1<u_b.size()))
                                                {
                                                        part_sum+=u_b[i].p*u_b[i].N;
                                                        W_n-=u_b[i].N;
                                                        i++;
                                                }
                                                temp+=ub[a][k]*part_sum;
                                }
                        }
                        //cout<<temp<<endl;
                        //cout<<"G "<<ub_sum<<endl;
                        result[b]=min(m[b],temp);
                        next_ub[a][b]=result[b];
                        //next_m[b] = max(next_m[b], result[b]);
                }
        }

        for(int i=0;i<256;i++)
        {
                //m[i]=next_m[i];
                for(int j=0;j<256;j++)
                        ub[i][j]=next_ub[i][j];
        }
}
```

```
struct my_a
{
        uint8_t a1;
        uint8_t a2;
};

my_a pre_k[65536], pre_b[65536];


void pre_k_b()
{
        for (int i = 0;i < 65536;i++)
        {
                uint8_t a1_t, a2_t, a3_t, a4_t, b1, b2, a1, a2;
                a1_t = i % 16;
                a2_t = (i >> 4) % 16;
                a3_t = (i >> 8) % 16;
                a4_t = (i >> 12) % 16;
                b1 = i % 256;
                b2 = (i >> 8) % 256;
                a1 = a1_t + (a4_t << 4);
                a2 = a3_t + (a2_t << 4);
                pre_k[i].a1 = a1;
                pre_k[i].a2 = a2;
                pre_b[i].a1 = b1;
                pre_b[i].a2 = b2;
                big_w[i] = get_weight(i);
        }
}

void pre_W_k_b()
{
        for (int i = 0;i < 65536;i++)
        {
                for (int j = 0;j < 65536;j++)
                {
                        if (((Wint64[pre_k[j].a1][pre_b[i].a1]) > 0) && ((Wint64[pre_k[j].a2][pre_b[i].a2]) > 0))
                        {
                                W_k_b[i].push_back(j);
                        }
                }
        }
}



void UB2() // 8/256 = 2^(-5) and 2^(B-1) B=9  (big_w[b] == wt(b))
{

                for (int i = 0;i < 256;i++)
                {
                        m[i] = 0;
                }
                long double result[256];

        for(int i=0;i<256;i++)
        {
                ub[i][0]=0;
                ub[0][i]=0;
        }

        ub[0][0]=1;
        for(int a=1;a<256;a++){
                result[0]=0;
                FILE* file = fopen("UB2.dat", "ab");
                for(int b=1;b<256;b++){
                        result[b]=0;
                        if ((big_w[a] + big_w[b]) > 8) {

                                        uint64_t big_w1b = (1 << big_w[b]);
                                        uint64_t big_w1a = (1 << big_w[a]);
                                        uint64_t  t2 = 1;
                                        t2 <<= 40;
                                        double t1a = (1 / (1.0 * big_w1a));
                                        t1a /= big_w1a;
                                        t1a *= t1a;
                                        t1a /= big_w1a;
                                        double t1b = (1 / (1.0 * big_w1b));
                                        t1b /= big_w1b;
                                        t1b *= t1b;
                                        t1b /= big_w1b;
                                        double t2p = (1 / (t2 * 1.0));
                                        result[b]=min(t1b, min(t1a, t2p));
                                        ub[a][b]=result[b];
                                }
                                m[b] = max(m[b], result[b]);
                        }
                                        fwrite(result, sizeof(double), 256, file);
                                        fclose(file);
                                        cout << a << endl;
                        }


                FILE* m_status = fopen("M2.dat", "wb");
                fwrite(m, sizeof(double), 256, m_status);
                fclose(m_status);
        }
```

```
/*
	pre_k[k] =
*/

void UB3()
{
	double next_m[256];
	for (int i = 0;i < 256;i++)
	{
		next_m[i] = 0;
	}
	/*FILE* m_pre = fopen("M4.dat", "rb");
	fread(m,sizeof(double), 65536, m_pre);
	fclose(m_pre);

	FILE* ub_pre = fopen("UB4_3.dat", "rb");

*/
	/*
	FILE* ub_pret = fopen("UB5_2.dat", "rb");

	for(int i=0;i<12711;i++)
	{
		FILE* file = fopen("UB5_2t.dat", "ab");
		fread(ub, sizeof(double), 65536, ub_pret);
		for(int b=0;b<65536;b++)
		{
			next_m[b] = max(next_m[b], ub[b]);
		}
		fwrite(ub, sizeof(double), 65536, file);
		fclose(file);
		cout<<i<<endl;
	}
	fclose(ub_pret);*/

	for (int a = 0;a < 256;a++)
	{
		double result[256];
		//fread(ub, sizeof(double), 65536, ub_pre);

		//FILE* file = fopen("UB5_3.dat", "ab");

		for (int b = 0;b < 256;b++)
		{

			double t;

			int big_wIb = (1 << big_w[b]);
			double t1b = (1 / (1.0 * big_wIb));
			                                                t1b /= big_wIb;
			                                                t1b *= t1b;
			                                                t1b /= big_wIb;
			int big_wIa = (1 << big_w[a]);
			double t1a = (1 / (1.0 * big_wIa));
			                            t1a /= big_wIa;
			                            t1a *= t1a;
			                            t1a /= big_wIa;
			result[b] = 0;
			//for(int i=0;i<65535;i++)
				//ub[i]=1;

			for (unsigned int k = 0;k < 256;k++)
			{
				t = Wint64[k][b] * ub[a][b] * t1b * t1a;
				//t *= Wint64[pre_k[W_k_b[b][k]].a2][pre_b[b].a2];
				result[b] += t;
			}
			//result[b] *= t1b; // overflow

			//cout<<result[b]<<" "<<t1b<<endl;
			result[b] = min(m[b], result[b]);
			ub[a][b]=result[b];
			next_m[b] = max(next_m[b], result[b]);
			//char k;
			//cin>>k;
		}
		//fwrite(result, sizeof(double), 65536, file);
		//fclose(file);
		//cout << a << endl;
	}
	for(int i=0;i<256;i++)
		m[i]=next_m[i];
	//fclose(ub_pre);
	//FILE* m_status = fopen("M5_3.dat", "wb");
	//fwrite(next_m, sizeof(double), 65536, m_status);
	//fclose(m_status);
}

void UB4()
{
	double m[65536],next_m[65536],ub[65536];
	for (int i = 0;i < 65536;i++)
	{
		next_m[i] = -1488;
	}
	FILE* m_pre = fopen("M3p.dat", "rb");
	fread(m,sizeof(double), 65536, m_pre);
	fclose(m_pre);
```

```
        FILE* ub_pre = fopen("UB3p.dat", "rb");

        for (int a = 0;a < 65536;a++)
        {
                double result[65536];
                fread(ub, sizeof(double), 65535, ub_pre);

                FILE* file = fopen("UB4p.dat", "ab");

                for (int b = 0;b < 65536;b++)
                {

                        double t;

                        int big_wlb = (1 << big_w[b]);
                        double t1b = (1 / (1.0 * big_wlb));
                                                                t1b /= big_wlb;
                                                                t1b /= t1b;
                        result[b] = 0;


                        for (unsigned int k = 0;k < W_k_b[b].size();k++)
                        {
                                t = Wint64[pre_k[W_k_b[b][k]].a1][pre_b[b].a1] * ub[W_k_b[b][k]];
                                t *= Wint64[pre_k[W_k_b[b][k]].a2][pre_b[b].a2] * t1b;
                                result[b] += t;
                        }
                        result[b] = min(m[b], result[b]);
                        next_m[b] = max(next_m[b], result[b]);
                }
                fwrite(result, sizeof(double), 65536, file);
                fclose(file);
                cout << a << endl;
        }
        fclose(ub_pre);
        FILE* m_status = fopen("M4p.dat", "wb");
        fwrite(next_m, sizeof(double), 65536, m_status);
        fclose(m_status);
}




int main()
{

        //int count_zero=0,count_nz=0;
        for (int a = 0;a < 16;a++)
        {
                for (int b = 0;b < 16;b++)
                {
                        Wint64[a][b] = get_sumWn(a, b);
                }
        }

        cout << "W done" << endl;



        //pre_k_b();
        //cout << "PRE done" << endl;


        //string t;
        //t.compare()
        //pre_W_k_b();
        //cout << "PRE W_K_B" << endl;
        //UB3();

        /*double m1[65536],m2[65536],m3[65536],m4[65536],m[65536];
        FILE* m1f = fopen("M4_1.dat", "rb");
        FILE* m2f = fopen("M4_2.dat", "rb");
        FILE* m3f = fopen("M4_3.dat", "rb");
        FILE* m4f = fopen("M4_4.dat", "rb");
        FILE* mf = fopen("M4.dat", "wb");
        fread(m1,sizeof(double), 65536, m1f);
        fread(m2,sizeof(double), 65536, m2f);
        fread(m3,sizeof(double), 65536, m3f);
        fread(m4,sizeof(double), 65536, m4f);
        for(int i=0;i<65536;i++)
        {
                m[i]=max(m1[i],max(m2[i],max(m3[i],m4[i])));
        }
        fwrite(m, sizeof(double), 65536, mf);
        fclose(mf);
        fclose(m1f);
        fclose(m2f);
        fclose(m3f);
        fclose(m4f);*/
        cout<<"A"<<endl;
        for (int j = 0;j < 4;j++)
                for (int a = 0;a < 256;a++)
                {
                        for (int b = 0;b < 256;b++)
                        {
                                ds0[a][b][j] = ds(a, b, j);
                        }

                }
```

```
long double QD_arr[256][2][4];
for(int k=0;k<4;k++)
{

	for(int a=1;a<256;a++)
	{
		long double tmp=0;
		for(int c=0;c<256;c++)
		{
			long double d=ds0[a][c][k];
			for(int i=0;i<4;i++)
			{
				d*=ds0[a][c][k];
			}
			tmp+=d;

		}

		QD_arr[a][0][k]=tmp;
	}

	for(int b=1;b<256;b++)
	{
		long double tmp=0;
		for(int c=0;c<256;c++)
		{
			long double d=ds0[c][b][k];
			for(int i=0;i<4;i++)
			{
				d*=ds0[c][b][k];
			}
			tmp+=d;

		}
		QD_arr[b][1][k]=tmp;

	}
}



	long double QD=0;
			{
					long double t;
					for(int j=1;j<256;j++)
					{
						t=max(QD_arr[j][0][1],QD_arr[j][1][1]);
						//cout<<log2(t)<<endl;
						QD=max(t,QD);
					}
			}


////
cout<<"B"<<endl;
//////sort
for (int k = 0;k < 4;k++)
{
	for (int a = 1;a < 256;a++)
	{
		for (int i = 1;i < 256;i++)
			for (int j = 1;j < 256;j++)
			{
				if (ds0[a][i][k] > ds0[a][j][k])
				{
					long double t = ds0[a][i][k];
					ds0[a][i][k] = ds0[a][j][k];
					ds0[a][j][k] = t;
				}
			}
	}
}
cout<<"C"<<endl;
////
for (int k = 0;k < 4;k++)
{
	for (int b = 1;b < 256;b++)
	{
		for (int i = 1;i < 256;i++)
			for (int j = 1;j < 256;j++)
			{
				if (ds0[i][b][k] > ds0[j][b][k])
				{
					long double t = ds0[i][b][k];
					ds0[j][b][k] = ds0[i][b][k];
					ds0[j][b][k] = t;
				}
			}
	}
}
////
cout<<"D"<<endl;
uint64_t counter = 1;
long double cur = -1;
bool first = 1;
MyPair t;
//////count
for (int k = 0;k < 4;k++)
{
```

```
        {
                for (int b = 1;b < 256;b++)
                {
                        if (cur == ds0[1][b][k]) counter++;
                        else
                        {
                                if ((!first) && (cur)) {
                                        t.p = cur;
                                        t.N = counter;
                                        dpN[k].push_back(t);
////
                                }
                                else first = 0;
                                cur = ds0[1][b][k];
                                counter = 1;
                        }
                }
        }
}

MyPair u_0;
u_0.N = 1; u_0.p = 1;
U[0].push_back(u_0);
cout<<"E"<<endl;
for (int i = 1;i < 16;i++)
{
        U[i] = u(i);
}
vector<MyPair> tmp;
ofstream ofst("U256.txt");
cout<<"G"<<endl;
for (int i = 0;i < 16;i++)
{
        for (int j = 0;j < 16;j++)
        {
                tmp = compose(U[i], U[j]);
                U256[i*16+j]=tmp;
                ofst << "*" << i << "␣␣" << j << endl;
                for (unsigned int j = 0;j < tmp.size();j++)
                {
                        ofst << tmp[j].p << "␣" << tmp[j].N << endl;
                }
        }
        cout << i << endl;
}
ofst.close();
cout<<"H"<<endl;


UB_42_2(QD);
ofstream mm2("M2.txt");
for(int j=0;j<256;j++)
                        mm2<<m[j]<<"␣";
mm2.close();
ofstream ofs("UB2.txt");
for(int i=0;i<256;i++)
{
        for(int j=0;j<256;j++)
        {
                ofs<<log2(ub[i][j])<<"␣";
        }
        ofs<<endl;
}
ofs.close();
long double maxx=0, inf = 3.50345e-300;


int count_zero=0;
for(int a=1;a<256;a++)
        for(int b=1;b<256;b++)
                {
                        maxx=max(maxx,ub[a][b]);
                if(ub[a][b]<inf)
                        {
                                count_zero++;
                        }
                }


cout<<"Raund␣2␣--␣2^"<< log2(maxx)<<"␣␣Zero␣--␣"<<count_zero<<endl;
count_zero=0;
maxx=0;
for(int i=0;i<7;i++)
{
        UB4_2_3();
        maxx=0;
        for(int a=1;a<256;a++)
                for(int b=1;b<256;b++)
                {
                                        maxx=max(maxx,ub[a][b]);
                                        if(ub[a][b]<inf)
                                                {
                                                        count_zero++;
                                                }
                                        }
        cout<<"Raund␣"<<i+3<<"␣--␣2^"<< log2(maxx)<<"␣␣Zero␣--␣"<<count_zero<<endl;
        count_zero=0;
}
ofstream ofs1("UB10.txt");
for(int i=0;i<256;i++)
{
```

```
        for(int j=0;j<256;j++)
        {
                ofs1<<log2(ub[i][j])<<" ";
        }
        ofs1<<endl;
}

ofs1.close();
ofstream mm10("M10.txt");
        for(int j=0;j<256;j++)
                                mm10<<m[j]<<" ";
        mm10.close();



////
////ofs1.close();
////
////ofstream ofs2("U65535.txt");
////vector<vector<MyPair>> U65535;
////for (int i = 0; i < 256; i++)
////{
////    for (int j = 0; j < 256; j++)
////    {
////            tmp = compose(U256[i], U256[j]);
////            U65535.push_back(tmp);
////            ofs2 << "*" << i << "  " << j << endl;
////            for (int j = 0; j < tmp.size(); j++)
////            {
////                    ofs2 << tmp[j].p << " " << tmp[j].N << endl;
////            }

////
////    }
////    cout << i << endl;
////}
////ofs2.close();
//cout << UB(65535, 65535)<<endl;

////ofstream ub2("UB2.txt");
///*ofstream m2("M2.txt");
//int m[65536];
//for (int i = 0; i < 65536; i++)
//      m[i] = -1488;

//for (int j = 0; j < 65536; j++)
//      for (int i = 0; i < 65536; i++)
//      {
//              m[j] = max(m[j], UB2(i, j));
//      }

//ub2 << endl;
//for (int i = 0; i < 65536; i++)
//{
//      for (int j = 0; j < 65536; j++)
//      {
//              ub2 << UB2(i, j)<<" ";
//      }
//      ub2 << endl;
//      cout << i << endl;
//}
//int tru_m = -1488;
//for (int i = 0; i < 65536; i++)
//{

//      m2 << m[i] << " ";
//}
//m2 << endl;
//m2.close();
////ofstream W_f("W_full2.txt");

////
////for (int b = 32768; b < 65536; b++)
////{
////    for (int a = 0; a < 65536; a++)
////    {
////            BigInteger W_full;
////            //__int128 t=0;
////            W_full = W[pre_k[a].a1][pre_b[b].a1] * W[pre_k[a].a2][pre_b[b].a2];
////            if (W_full > 0)
////            {
////                    W_f << std::setprecision(9) << log2(W_full) << " ";
////            }
////            else W_f << "-1" << " ";
////    }
////    W_f << endl;
////    cout << b << endl;
////}
////W_f.close();
//ofstream m3f("M3.txt");

//cout << "M2 done" << endl;



////ofstream ub3("UB3.txt");
////ofstream m3f("M3.txt");

////long double m3[65536], t_ub;
////int i = 0;
////for (int j = 0; j < 65536; j++)
```

```
        ////{
        ////      t_ub = min(m[j]*1.0, UB3(i, j));
        ////      m3[j] = t_ub;
        ////      ub3 << t_ub << " ";
        ////}
        ////ub3 << endl;
        ////cout << "UB3 i=0" << endl;
        ////for (int i = 1;i < 65536;i++)
        ////{
        ////      for (int j = 0;j < 65536;j++)
        ////      {
        ////              t_ub = min(m[j],UB3(i, j));
        ////              ub3 << t_ub << " ";
        ////              m3[j] = max(m3[j-1], t_ub);
        ////      }
        ////      ub3 << endl;
        ////      cout << i << endl;
        ////}
        ////ub3.close();

        ////for (int i = 0;i < 65536;i++)
        ////{
        ////      m3f << m3[i] << " ";
        ////}
        ////m3f << endl;
        ////m3f.close();


        //ub2.close();
        //ofs.close();
        //system("PAUSE");
    return 0;
}
```