МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт искусственного интеллекта
Базовая кафедра №252 – информационной безопасности

РАБОТА ДОПУЩЕНА К ЗАЩИТЕ
Заведующий кафедрой

_____
(подпись)

Корольков А.В.
(Фамилия И.О.)

«___» _____ 2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
по специальности 10.05.01 «Компьютерная безопасность»

На тему Атака со связанными ключами на один из шифров сети Фейстеля

Обучающийся _____ Исаев Кади Магомедович

Шифр 18К0188
Группа ККСО-01-18

Руководитель работы: _____ старший преподаватель Кирюхин В.А.

Москва 2024

Оглавление

Введение

Оценка устойчивости блочных шифров относительно дифференциального метода криптоанализа уже давно стала обязательным и привычным требованием доказательства их криптостойкости. Однако, такие оценки в подавляющем большинстве случаев проводятся в классической модели дифференциального криптоанализа с общим ключом. Криптоаналитиками со всего мира был написан целый пласт работ в таком жанре: [5], [10], [16], [18], [22], [17], [12], [14]. Атаки в таком стиле, хоть и являются наиболее вероятными с точки зрения их применимости в реальных условиях, не являются единственными. Гораздо меньше работ посвящено дифференциальному криптоанализу блочных шифров в модели связанных ключей. Такой жанр атак хоть и получает меньше внимания, но оценка устойчивости к атакам на связанных ключах тоже стала важным аспектом доказательства криптостойкости блочных шифров. Было выпущено множество статей с криптоанализом в таком стиле: [4], [6], [13], [2]. Атаки на связанных ключах могут быть улучшены, комбинируя их с другими стилями атак, например, методом бумеранга и его усовершенствованными версиями и техниками [23, 9].

Целью данной работы является разработка атак на блочный шифр Khazad с в модели связанных ключей.

В первой главе будут рассмотрены основные термины и техники, связанные с дифференциальным криптоанализом, описание алгоритма шифрования Khazad, к которому и будут применены упомянутые ранее определения.

Во второй главе будут описаны пути развития идей разностного криптоанализа, а именно классическая атака методом бумеранга и особенности построения атак в модели связанных ключей. Эти знания необходимы для их комбинированного использования при построении атак в следующих главах.

Используя сведения из предыдущих глав, в третьей главе представлена атака методом бумеранга в модели связанных ключей на 4 раунда шифра Khazad с модифицированной ключевой развёрткой, её оценка и программная реализация, подтверждающая корректность алгоритма и его сложность.

В четвертой главе предложена теоретическая атака методом бумеранга в модели связанных ключей на 5 раундов шифра Khazad с модифицированной клю-

чевой развёрткой, которая при её комбинации с уже давно известной атакой [19] позволяет существенно уменьшить сложность последней.

В заключении приведены результаты, полученные в работе, и указано перспективное направление для дальнейшего исследования по данной теме.

ГЛАВА 1

О дифференциальном криптоанализе

Дифференциальный, также известный как разностный, криптоанализ - один из самых полезных инструментов в арсенале любого криптоаналитика. Впервые в публичном поле эта техника криптоанализа была освещена Эли Бихамом и Ади Шамиром в [5]. С тех пор появилось множество стилей криптографических атак, фундаментом которых является дифференциальный криптоанализ. В данном разделе будет представлено описание оригинального дифференциального криптоанализа. В последующих разделах будет представлено развитие данной идеи в более новых техниках криптографических атак.

1.1 Используемые обозначения и определения

Будем обозначать $\mathbb{F}_n$ - конечное поле из n элементов, $\mathbb{F}_n^m$ - векторное пространство размерности m над конечным полем из n элементов. Некоторые обозначения будут вводиться в процессе повествования в соответствующих разделах.

Определение 1. Бит - элемент конечного поля из двух элементов.

$$bit \in \mathbb{F}_2$$

Определение 2. Байт - элемент векторного пространства размерности 8 над конечным полем из двух элементов.

$$byte \in \mathbb{F}_2^8$$

Определение 3. Блок - элемент $\mathbb{F}_n^m, n, m \in \mathbb{Z}$.

Определение 4. Ключ - секретный параметр алгоритма шифрования.

Определение 5. Шифр - обратимое преобразование, зависящее от ключа, используемое для трансформации, сокрывающей исходный смысл информации.

Определение 6. Блочный шифр - шифр, осуществляющий преобразование над блоками информации определённой длины.

Определение 7. Открытый текст - защищаемая, несокрытая, информация.

Определение 8. Закрытый текст, шифротекст - защищённая, сокрытая, информация.

Определение 9. Подстановочно-перестановочная сеть - дизайн алгоритмов блочного шифрования, использующих в своей структуре операции подстановки и перестановки, которые согласно Клоду Шеннону [20] должны использоваться в шифре, обеспечивающем хорошую стойкость.

1.2 Необходимые понятия из дифференциального криптоанализа

В работе будем рассматривать дифференциальный криптоанализ в рамках его применения к блочным шифрам, а точнее к подстановочно-перестановочным сетям. Опишем только те понятия, которые будут использованы для построения атак, выводить их будем на примере шифра Khazad [1], за одно опишем и его.

Определение 10. Разностью будем называть сложение по модулю два между двумя байтами текста $x, x^{'} \in \mathbb{F}_{2^8}$.

$$\Delta x = x \oplus x^{'}$$

Суть дифференциального метода состоит в анализе метаморфоз, происходящих с такими разностями при прохождении через преобразования криптоалгоритма.

Определение 11. Переход одной разности в другую под влиянием одного или нескольких последовательных абстрактных преобразований $func_i, i \in \overline{1,n}$ криптоалгоритма будем называть дифференциальным путём и обозначать $\Delta x \xrightarrow{func_1} \ldots \xrightarrow{func_n} \Delta y$.

Определение 12. Совокупность дифференциальных путей из фиксированной разности Δx в фиксированную разность Δy будем называть дифференциалом.

1.2.1 Описание шифра Khazad

Шифр Khazad был разработан в 1995 году с участием бельгийского криптографа Винсента Реймена, широко известного как соавтор самого используемого алгоритма шифрования в мире - AES.
По своей структуре криптоалгоритм Khazad является подстановочно-перестановочной сетью, обладает инволюционной структурой и имеет следующие характеристики:

- Размер блока - 64 бита
- Размер ключа - 128 бит
- Стандартное число раундов - $R = 8$

Опишем структуру преобразований, применяемых в каждом раунде алгоритма и развёртки ключа, в соответствии со спецификацией шифра [1]. При этом при описании изменим принятую в спецификации шифра нотацию для обозначения операций на более интуитивную нотацию из [15]:

- S - нелинейное преобразование
- L - линейное преобразование
- $X[K^r]$ - сложение с раундовым ключом K^r по модулю 2

Процедура зашифрования:

$$C = X[K^R] \circ S \circ \left(\bigcirc_{r=1}^{R-1} X[K^r] L S \right) \circ X[K^0](P)$$

Процедура расшифрования:

$$P = X[\widetilde{K^R}] \circ S \circ \left(\bigcirc_{r=1}^{R-1} X[\widetilde{K^r}] L S \right) \circ X[\widetilde{K^0}](C),$$

где $\widetilde{K^0} = K^R, \widetilde{K^R} = K^0, \widetilde{K^r} = L(K^{R-r}), r \in \overline{1, R-1}$. Процедура расшифрования отличается от процедуры зашифрования только используемыми раундовыми ключами, достигается это благодаря инволюционной природе преобразований криптоалгоритма.

Операции умножения в преобразованиях шифра производятся в кольце многочленов по модулю $GF(2)/p(x)$, где $p(x) = x^8+x^4+x^3+x^2+1$ - неприводимый. Обозначать данное поле будем как $GF(2^8)$, соответственно восьмибайтовый блок будет обозначаться как $GF(2^8)^8$.

1.2.2 Преобразование S и его криптографические свойства

Описание преобразования S

Функция S в качестве аргумента принимает восьмибайтную строку и применяет подстановку ко всем отдельным байтам аргумента независимо:

$$S : GF(2^8)^8 \to GF(2^8)^8$$

$$S(a) = b \Leftrightarrow b[i] = Sbox[a[i]], i \in \overline{0,7}$$

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

Подстановка была выбрана таким образом, чтобы отображение, которое получается при её применении, было обратно самому себе, то есть являлось инволюцией:

$$Sbox[Sbox[x]] = x, x \in \mathbb{F}_2^8$$

Криптографические свойства S

Изучим свойства байтовой подстановки Khazad'a с помощью техник разностного криптоанализа.

Пусть даны два байта открытого текста $x, x^{'} \in \mathbb{F}_2^8$, с разностью между ними равной Δx. Необходимо ответить на вопрос: каким образом байтовая подстановка влияет на Δx?

$$S(x) \oplus S(x^{'}) = S(x) \oplus S(x \oplus \Delta x) = y \oplus y^{'} = \Delta y \tag{1.1}$$

Sbox Khazad'a биективен по отношению к конкретным значениям, то есть однозначно отображает один байт в другой, однако по отношению к разностям он не инъективен, то есть одна разность может отображаться в какое-то число других разностей. Здесь естественным образом возникает понятие таблицы распределения разностей.

Определение 13. Таблица распределения разностей - квадратная матрица DDT, размером $2^8 \times 2^8$ элементы которой равны числу решений уравнения 1.1 относительно x при всевозможных переходах $\Delta x \xrightarrow{sbox} \Delta y$:

$$DDT[\Delta x][\Delta y] = |\{x : S(x) \oplus S(x \oplus \Delta x) = \Delta y\}|$$

Если предположить, что переменная $x \in \mathbb{F}_2^8$ распределена равновероятно и независимо, то имеет место распределение вероятностей числа решений уравнения 1.1 следующего вида:

$$\begin{pmatrix} 0 & 2 & 4 & 6 & 8 & 256 \\ \frac{39999}{2^{16}} & \frac{19531}{2^{16}} & \frac{5013}{2^{16}} & \frac{889}{2^{16}} & \frac{104}{2^{16}} & \frac{1}{2^{16}} \end{pmatrix}$$

$$Pr(|\{x : S(x) \oplus S(x \oplus \Delta x) = \Delta y\}| > 0) = 0.389679$$

- вероятность того, что случайно выбранная ячейка таблицы распределения разностей будет ненулевой.

Математическое ожидание такого распределения равно:

$$0 \cdot \frac{39999}{2^{16}} + 2 \cdot \frac{19531}{2^{16}} + 4 \cdot \frac{5013}{2^{16}} + 6 \cdot \frac{889}{2^{16}} + 8 \cdot \frac{104}{2^{16}} + 256 \cdot \frac{1}{2^{16}} = 1$$

Это знание нам будет необходимо для проведения практических экспериментов. Стоит понимать ”физический смысл” данного математического ожидания. Для конкретных значений Δx и Δy наиболее вероятна ситуация, когда мы не найдём ни одного решения, но некоторые такие пары разностей имеют несколько решений, следовательно при большой выборке можно ожидать, что по отношению к общему числу уравнений для каждого будет найдено одно решение.

1.2.3 Преобразование L и его криптографические свойства

Описание преобразования L

Функция линейного преобразования представляет собой умножение аргумента функции в конечном поле слева на матрицу H, выбранную так, чтобы отображение, которое получается при таком умножении было так же инволюцией, поэтому матрица симметричная и унитарная.

$$L : GF(2^8)^8 \to GF(2^8)^8$$

$$L(x) = y \Leftrightarrow y = x \cdot H.$$

$$H = \begin{bmatrix} 1 & 3 & 4 & 5 & 6 & 8 & B & 7 \\ 3 & 1 & 5 & 4 & 8 & 6 & 7 & B \\ 4 & 5 & 1 & 3 & B & 7 & 6 & 8 \\ 5 & 4 & 3 & 1 & 7 & B & 8 & 6 \\ 6 & 8 & B & 7 & 1 & 3 & 4 & 5 \\ 8 & 6 & 7 & B & 3 & 1 & 5 & 4 \\ B & 7 & 6 & 8 & 4 & 5 & 1 & 3 \\ 7 & B & 8 & 6 & 5 & 4 & 3 & 1 \end{bmatrix} \tag{1.2}$$

Криптографические свойства L

Линейное преобразование детерминированно отображает одну разность в другую:

$$L(x) \oplus L(x \oplus \Delta x) = L(x) \oplus L(x) \oplus L(\Delta x) = L(\Delta x) = \Delta y.$$

Назначение линейного преобразования заключается в том, что такое преобразование производит рассеивание, то есть активный байт состояния до линейного преобразования должен активировать как можно большее число байтов после линейного преобразования. В большинстве современных блочных шифров это достигается с использованием матриц МДР кодов. Степень рассеивания линейного преобразования характеризуется метрикой, именуемой разностным коэффициентом рассеивания:

$$\mathcal{B}(L) \equiv \min_{A \neq 0} \{w_h(A) + w_h(L(A))\},$$

где $A \in \mathbb{F}_{2^8}^8$, w_h - вес Хэмминга. Для линейного преобразования Khazad'a $\mathcal{B}(L) = 9$, что значит, что если на вход линейного преобразования подаётся разность, активная в одном байте, то на выходе активными будут все восемь байтов. При построении атак это свойство будет активно использоваться.

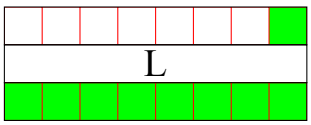

Рисунок 1 – Прохождение однобайтовой разности через линейное преобразование.

В дальнейшем будем обозначать белым цветом - неактивные байты разности, зелёным - активные байты, о потенциальных значениях которых мы знаем, синим - совпадающие разности после действия s-блока, чёрным - активные байты, чьи значения мы предсказать не можем.

1.2.4 Известные результаты криптоанализа шифра Khazad

Атака	Ключи	Раундов	Время	Память	Данные	Статья
Интегральная		3	2^{16}	2^{9}	2^{8}	[1]
Интегральная		4	2^{16+64}	2^{9}	2^{8}	[1]
Интегральная		5	2^{91}	2^{11}	2^{64}	[19]
Интегральная		5	2^{91}	2^{11}	2^{64}	[19]
Инвариантные пр-ва	Слабые	5	2^{35}		2^{32}	[11]
Инвариантные пр-ва	Слабые	6	2^{43}		2^{32}	[11]
Алгебраическая		6	2^{90}	2^{68}	2^{64}	[8]
Бумеранговая	Слабые Связанные	7	2^{50}		2^{50}	[9]

Таблица 1 – Атаки на шифр Khazad.

Стоит отдельно отметить, что Khazad структурно очень похож на Кузнечик, однако у них есть ряд тонких различий и по этой причине отдельные методы криптоанализа для этих шифров ведут себя по-разному. Например, атака на Khazad, приведённая в работе [11] использует специфические для Khazad'а свойства композиции нелинейного и линейного преобразований вследствие чего было обнаружено наличие 14 новых классов слабых ключей и построены соответствующие атаки по восстановлению ключа с их использованием. Для Кузнечика же такой метод неприменим.

В то же время некоторые структурные отличия предотвращают Khazad от некоторых способов атак на Кузнечик. Например, тот факт, что части мастер-ключа оригинального Khazad'а не используются в самом шифровании, а также отсутствие линейного преобразования в последнем раунде позволяют шифру Khazad иметь большую устойчивость к методу, описанному в [15].

Однако есть и методы, дающие схожие результаты. В данной работе при построении атак будет использоваться метод бумеранга в модели связанных ключей, теоретическое описание для которого будет приведено в следующей главе. Подобный метод использовался в [23] против пяти раундов шифра Кузнечик для эффективного восстановления первой части мастер-ключа(раундового

ключа K^0) с трудоёмкостью 2^{34} по времени, 2^{33} по памяти, 2^{24} по данным и 2 связанных ключа.

ГЛАВА 2

Атаки методом бумеранга в модели связанных ключей

2.1 Классическая атака методом бумеранга

Атака методом бумеранга была впервые представлена в [21] Дэвидом Вагнером в качестве дальнейшего развития идеи дифференциального криптоанализа. Суть метода заключается в том, что криптоаналитик пытается построить два коротких высоковероятных дифференциальных пути, а не один длинный, как в оригинальном дифференциальном криптоанализе. Он декомпозирует атакуемый алгоритм на две части, два подшифра, верхний и нижний, так же принято называть и соответствующие дифференциальные пути. Пусть

$$\begin{aligned} E(P) &= E_2 \circ E_1(P), \\ Pr(\alpha \xrightarrow{E_1} \beta) &= p, \\ Pr(\gamma \xrightarrow{E_2} \delta) &= q, \end{aligned} \tag{2.1}$$

Подробная схема атаки:

1. Атакущий выбирает два блока открытого текста $P^1, P^2 \in \mathbb{F}_{2^8}^n$ таких, что $P^1 \oplus P^2 = \alpha$.

2. Далее производится зашифрование открытых текстов, получая пару шифротекстов $C^1 = E(P^1)$, $C^2 = E(P^2)$.

3. Атакующий получает ещё одну пару шифротекстов, добавляя к уже имеющейся паре известную заранее разность: $C^3 = C^1 \oplus \delta$, $C^4 = C^2 \oplus \delta$.

4. Новую пару шифротекстов пропускают через алгоритм расшифрования и получают ещё одну пару открытых текстов: $P^3 = E^{-1}(C^3)$, $P^4 = E^{-1}(C^4)$.

5. Новая пара открытых текстов должна иметь такую же разность, что и первая: $P^3 \oplus P^4 = \alpha$. Видим, что бумеранг вернулся.

Теперь проанализируем промежуточные состояния. Будем обозначать промежуточное состояние как $Q^i = E_0(P^i) = E_1^{-1}(C^i)$. Зная, что $Pr(Q^1 \oplus Q^2 = \beta) = p, Pr(Q^1 \oplus Q^3 = \gamma) = q, Pr(Q^2 \oplus Q^4 = \gamma) = q$, имеем:

$$\begin{aligned} Q^3 \oplus Q^4 = \; & E_0(P^3) \oplus E_0(P^4) = E_0(P^1) \oplus E_0(P^2) \\ & \oplus E_0(P^1) \oplus E_0(P^3) \oplus E_0(P^2) \oplus E_0(P^4) \\ & = (Q^1 \oplus Q^2) \oplus (Q^1 \oplus Q^3) \oplus (Q^2 \oplus Q^4) \\ & = \beta \oplus \gamma \oplus \gamma = \beta \end{aligned} \tag{2.2}$$

При полной независимости верхнего и нижнего дифференциальных путей друг от друга, что бывает не всегда, вероятность этого события равна:

$$Pr(Q^3 \oplus Q^4) = p^2 q^2. \tag{2.3}$$

Существует несколько техник, которые позволяют улучшить результат бумеранговой атаки. Одной из таких техник является *sbox switch*, описанный в [7].

Соединение через S-блок

Если последнее преобразование верхнего дифференциального пути обрабатывает байты состояния независимо друг от друга, как, например, это делает sbox, то при выполнении определенных условий между верхним и нижним дифференциальными путями возникает зависимость, которая позволяет сэкономить на вероятности перехода.

$$\begin{gathered} \Delta, \nabla \in \mathbb{F}_2^8 - \text{разности}, \\ Pr(\Delta \xrightarrow{sbox} \nabla) = \rho, \\ x_1 \oplus x_2 = \Delta, y_1 = sbox[x_1], y_2 = sbox[x_2], \\ y_1 \oplus y_2 = \beta = \nabla = \gamma, \\ y_3 = y_1 \oplus \nabla, y_4 = y_2 \oplus \nabla, \\ y_3 = y2, y4 = y1, \\ S^{-1}(y_1) \oplus S^{-1}(y_2) = S^{-1}(y_3) \oplus S^{-1}(y_4) = x_1 \oplus x_2 = \Delta. \end{gathered} \tag{2.4}$$

Это означает, что такая зависимость улучшит общую вероятность переключения с одного дифференциального пути на другой на ρ.

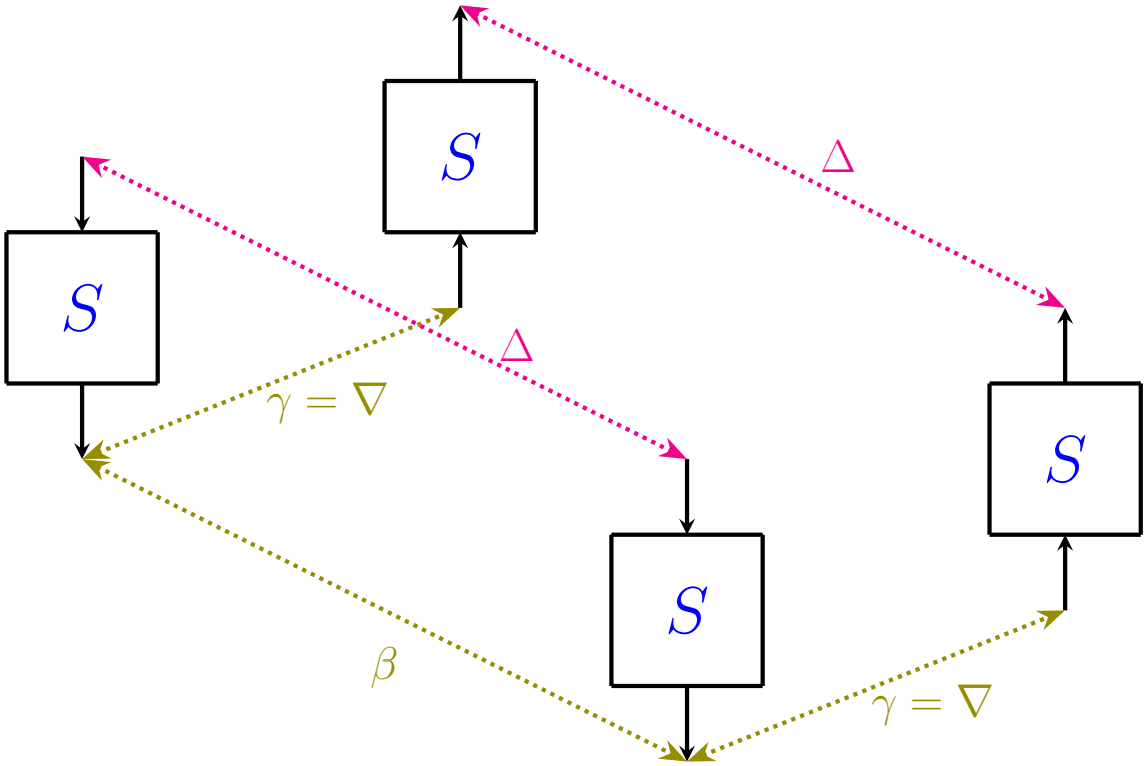

Рисунок 2 – Sbox соединение

2.2 Разработка атак в модели связанных ключей

В данном разделе будут исследоваться особенности построения атак на связанных ключах на блочный шифр. В качестве подопытного шифра будет использоваться шифр Khazad с модифицированной ключевой развёрткой.

2.2.1 Модель связанных ключей

Как мы могли заметить в классической модели дифференциального криптоанализа, добавление раундового ключа никак не влияет на число активных байтов состояния шифра. Об этом можно думать следующим образом: у противника имеется доступ к некоторому числу шифраторов, в каждом из которых используется один и тот же мастер-ключ. Криптоаналитик производит анализ дифференциальных путей только с тем учётом, что разность в таких атаках вносится только в открытые тексты. Что если предположить, что ключи шифраторов не совпадают и между ними имеется некоторое соотношение? В таком случае будем называть эти ключи связанными, они будут вносить дополнитель-

ную разность в соотношение между внутренними состояниями шифраторов. В общем случае это отношение может быть выражено какой-то определенной функцией, но в подавляющем большинстве случаев считают, что эта функция - простое сложение по модулю два.

Определение 14. Модель связанных ключей - тип криптографической атаки, при которой разность в состояние шифра может вноситься не только через открытые тексты, но и через соотношение между ключами.

Криптоаналитик в такой модели может не только знать о наличии определённого соотношения между ключами, но и выбирать это самое соотношение.

Итак, разности, выбранные криптоаналитиком, вносятся в мастер-ключ, являющийся аргументом функции ключевой развертки, которая в свою очередь генерирует из него какое-то количество раундовых ключей. Таким образом, разность, внесённая в мастер-ключ, начинает распространяться по раундовым ключам. Раундовые ключи вносят свою разность в состояние шифра на определенном такте его работы. Атакующему в такой модели приходится анализировать вероятности дифференциалов с учётом связанности ключей, то есть если в классической модели вероятность дифференциала определялась как:

$$Pr_{P,K}[E_K(P) \oplus E_K(P \oplus \Delta P) = \Delta C], \tag{2.5}$$

где на вероятность дифференциала влияют пары разностей ΔP и ΔC, то в модели связанных ключей она определяется так:

$$Pr_{P,K}[E_K(P) \oplus E_{K \oplus \Delta K}(P \oplus \Delta P) = \Delta C], \tag{2.6}$$

где на вероятность дифференциала влияют тройки разностей ΔP, ΔC и ΔK.

Изобразим какой-то дифференциальный путь в модели связанных ключей:

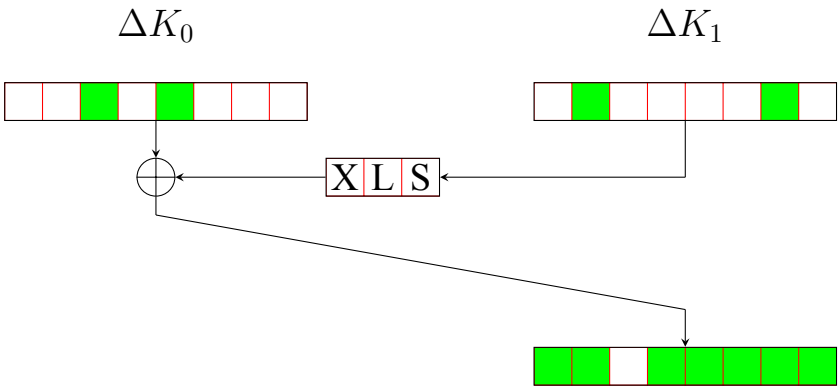

Рисунок 3 – Пример дифференциального пути на связанных ключах внутри сети Фейстеля ключевой развертки.

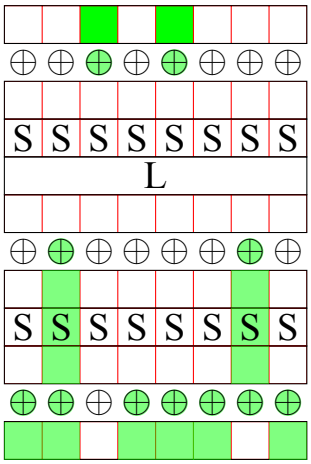

Рисунок 4 – Пример дифференциального пути на связанных ключах внутри шифрпреобразования.

На этих рисунках мы сразу же получили одну важную особенность такой модели. Можно заметить, что разности, вносимые при наложении раундовых ключей, накладываются на разности в состоянии шифрпреобразования.

Основной техникой при построении атак в модели связанных ключей является построение так называемых локальных коллизий. Понятие локальных коллизий берет своё начало в криптоанализе хэш-функций. Идея заключается в активизации байтов внутреннего состояния шифра внесением разности с последующим добавлением ключевой разности, которая наоборот должна эти байты деактивировать, что мы и могли заметить на приведенных выше схемах. Для построения подобных коллизий криптоаналитик пользуется схожестью функции развёртки ключа с самим алгоритмом шифрования.

В работе [1] авторами шифра Khazad утверждается, что схема ключевой развёртки унаследовала свойства раундовой функции шифра и была спроектирована таким образом, чтобы иметь отличные замешивающие свойства. В дальнейшем повествовании в главах, в которых будет приведено построение атак на Khazad, будет использоваться модифицированная ключевая развёртка. Точнее, алгоритм развёртки будет такой же, только в первых двух раундах шифрования будут использоваться части мастер-ключа в порядке их нумерации в спецификации, чего не происходит в оригинальном алгоритме. Это нужно для возможности реализации атаки на практике и проверки корректности наших рассуждений и оценок сложности на обычном персональном компьютере.

В итоге имеем:

$$K^0 = K^{-2},$$
$$K^1 = K^{-1}.$$

Остальные ключи генерируются сетью Фейстеля со структурой из оригинального алгоритма:

$$K^i = K^{i-2} \oplus LS(K^{i-1}), i \in \overline{2,5}.$$

2.3 Переход в модель связанных ключей в методе бумеранга

Атаки методом бумеранга на связанных ключах впервые были описаны в [3]. В схеме классической атаки методом бумеранга используется несколько шифраторов, использующих один и тот же ключ. Однако, как было выяснено, такая модель реализуется не всегда. Для того, чтобы осуществить переход для рассматриваемого метода в модель связанных ключей, между ключами должно существовать некое отношение. В таком случае возможны варианты. Например, все четыре возможных шифратора могут использовать разные ключи, которые при этом являются связанными, или два шифратора имеют связанные ключи, а другие два несвязанные. Соответственно, при этом верхний и нижний дифференциальные пути могут быть как на связанных ключах, так и на несвязанных. Переход в такую модель открывает большой простор для криптоаналитика по поиску хороших дифференциальных путей. В такой модели атакующий опро-

бует различные варианты ключевых разностей и анализирует их распространение в функции ключевой развёртки для каждого связанного дифференциального пути.

ГЛАВА 3

Атака методом бумеранга в модели связанных ключей на 4 раунда шифра Khazad с модифицированной ключевой развёрткой

Основополагающей работой для предлагаемой здесь атаки является [23].

Модель атаки: пусть противник имеет доступ к двум шифраторам. Он знает, или выбирает, соотношение между используемыми ключами. Сначала бумеранг летит на связанных ключах, а обратно возвращается на несвязанных.

Разобьём процесс шифрования на три части:

$$E(P) = E_2 \circ S \circ E_1(P)$$

$$E(P) = X[K^4]SX[K^3]L \circ S \circ X[K^2]LSX[K^1]LSX[K^0](P)$$

$$E_1 = X[K^2]LSX[K^1]LSX[K^0]$$

$$E_2 = X[K^4]SX[K^3]L$$

Теперь для двух подшифров E_1, E_2 необходимо построить высоковероятные дифференциальные пути.

3.1 Построение дифференциального пути внутри сети Фейстеля ключевой развёртки

Учитывая хорошие рассеивающие свойства линейного преобразования, выбор исходных ключевых разностей с малым числом активных байтов является довольно естественным желанием, соотносящимся со здравым смыслом. Выбираем разность исходных раундовых ключей следующим образом:

$$\Delta K_0 = 0$$
$$\Delta K_1 = \kappa$$

В таком случае, разность в ключах K_2 будет иметь вид:

$$\Delta K_2 \in LS(\kappa),$$

где мощность множества $|LS(\kappa)| < 2^8$.

Дифференциальный путь распространения разностей через сеть Фейстеля ключевой развертки будет выглядеть следующим образом:

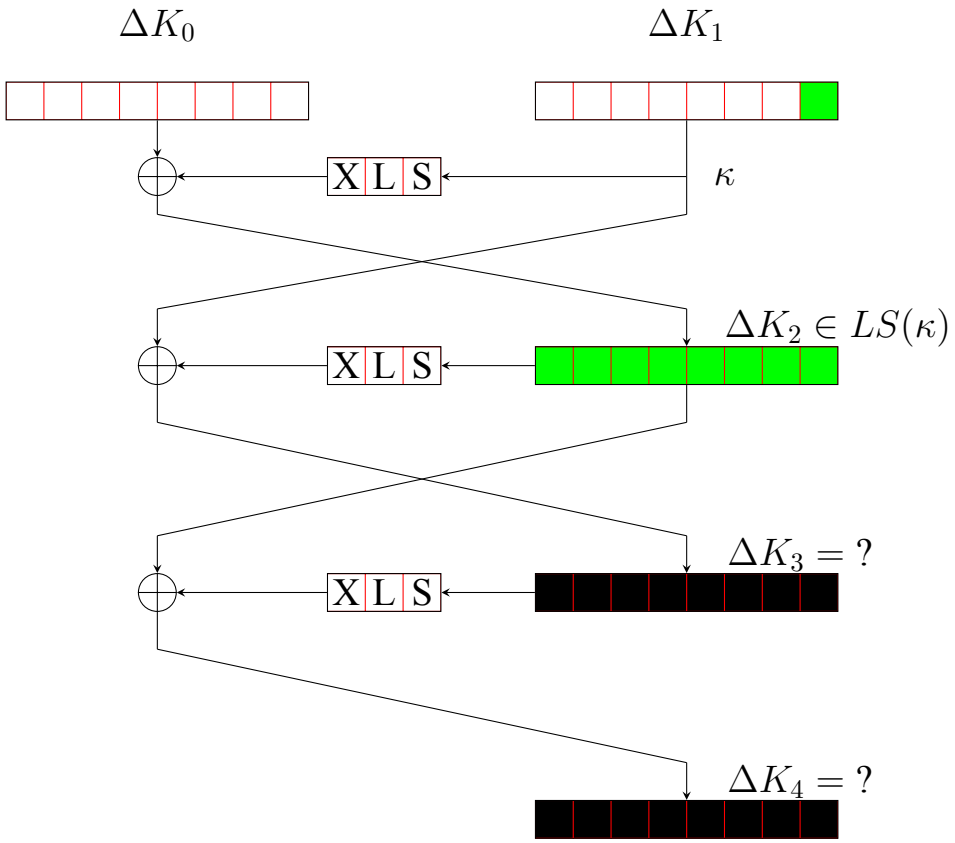

Рисунок 5 – Дифференциальный путь внутри сети Фейстеля ключевой развертки для атаки методом бумеранга на связанных ключах.

В итоге имеем следующий набор разностей:

$$\Delta K_0 = 0$$
$$\Delta K_1 = \kappa$$
$$\Delta K_2 = LS(\kappa)$$
$$\Delta K_3 = ?$$
$$\Delta K_4 = ?$$

3.2 Построение дифференциального пути на связанных ключах внутри шифрпреобразования

На вход шифраторов будем подавать один и тот же открытый текст, сгенерированный случайным образом $E_K(P) = Q$ и $E_{\tilde{K}}(P) = \tilde{Q}$. На выходе имеем пары шифротекстов, точную разность которых мы предугадать не можем.

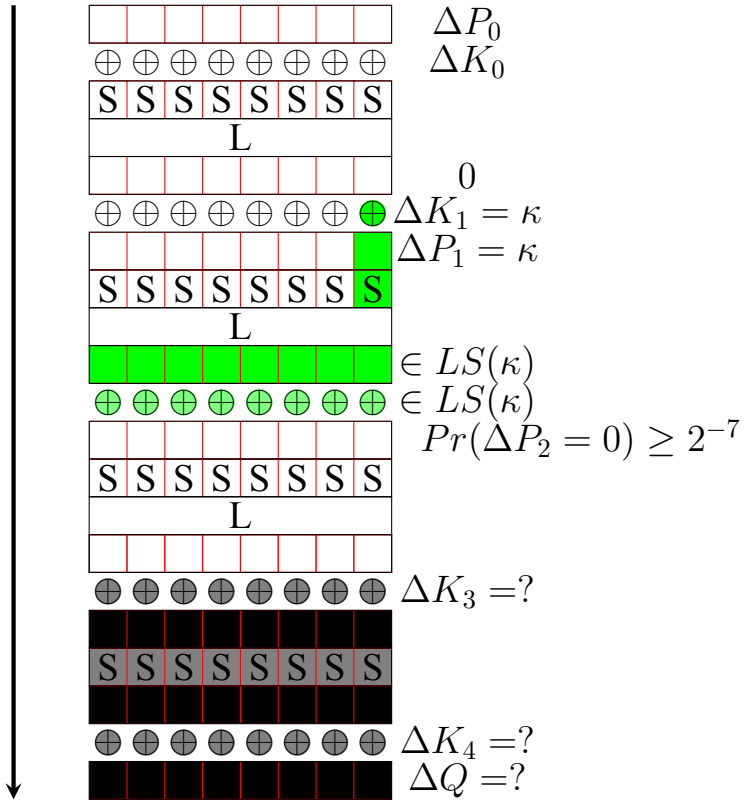

Рисунок 6 – Дифференциальный путь на связанных ключах внутри шифрпреобразования в направлении ”сверху вниз”.

$$\begin{gathered} Pr(X[K^2]LSX[K^1]LSX[K^0](P) \oplus \\ X[\widetilde{K^2}]LSX[\widetilde{K^1}]LSX[K^0](P) = 0) \geq 2^{-7} \end{gathered} \tag{3.1}$$

К каждому из полученных шифротекстов прибавляем разность δ отличающуюся только в одном байте $Q \oplus \delta = Z$ и $\tilde{Q} \oplus \delta = \tilde{Z}$ и идём вверх на несвязанных ключах.

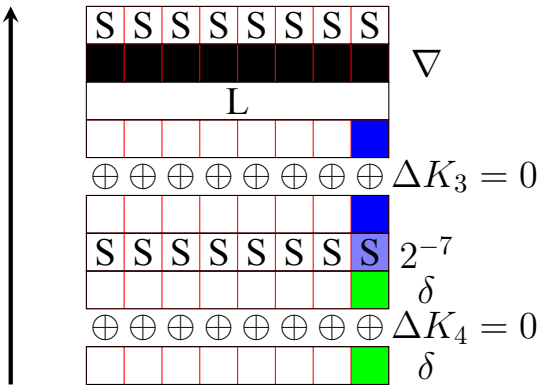

Рисунок 7 – Дифференциальный путь на несвязанных ключах внутри шифр-преобразования в направлении ”снизу вверх”.

$$\nabla = LX[K^3]SX[K^4](Z) \oplus LX[K^3]SX[K^4](Q),$$
$$\tilde{\nabla} = LX[K^3]SX[K^4](\tilde{Z}) \oplus LX[K^3]SX[K^4](\tilde{Q}),$$
$$Pr(\nabla = \tilde{\nabla}) \geq 2^{-7}.$$

Вероятность того, что зеленая разность под действием S-box’а перейдёт в синюю равна $p \geq 2^{-7}$. Далее детерминированно, то есть с вероятностью $p = 1$ ”перепрыгиваем” из дифференциального пути на несвязанных ключах в дифференциальный путь на связанных ключах в направлении ”вверх” с помощью соединения через S-box.

$$D = X[K^2]LSX[K^1]LSX[K^0](P),$$
$$Pr(S(S(D) \oplus \nabla) \oplus S(S(D \oplus 0) \oplus \nabla) = 0) = 1.$$

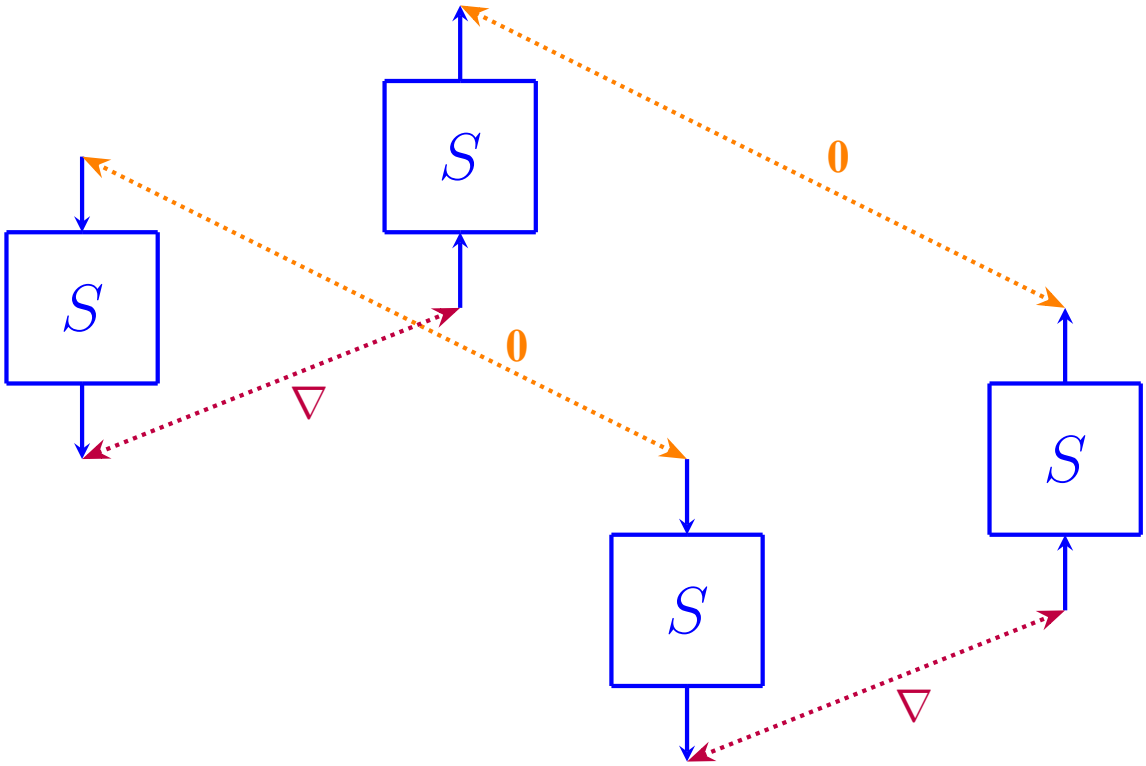

Рисунок 8 – Sbox соединение

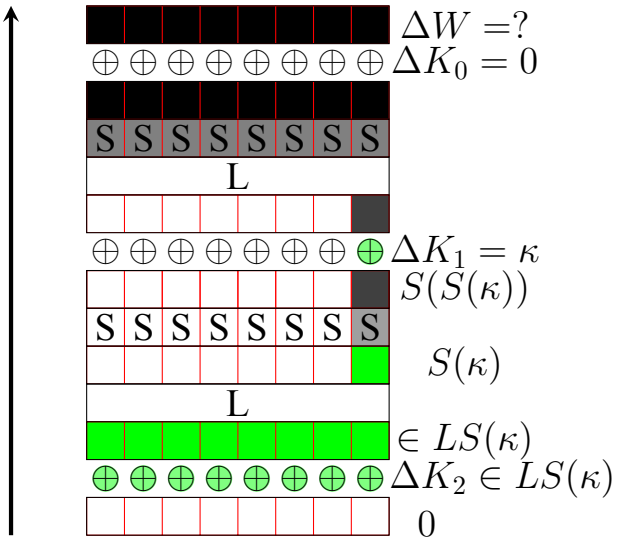

Рисунок 9 – Дифференциальный путь на связанных ключах внутри шифрпреобразования в направлении ”снизу вверх”.

Получили пару открытых текстов $W, \tilde{W}$, разность $\Delta W = W \oplus \tilde{W}$ между которыми мы знаем, но предугадать не можем.

$$Y = LSX[K^0](W)$$

$$\tilde{Y} = LSX[K^0](\tilde{W})$$

$$\Delta Y = Y \oplus \tilde{Y}$$

Вероятность того, что ΔY имеет один активный байт имеет нижнюю границу $p \geq 2^{-14}$.

Для каждой пары открытых текстов W и $\tilde{W}$ будем перебирать 255 возможных вариантов однобайтных разностей ΔY и решать уравнение вида:

$$S(W \oplus K^0) \oplus S(\tilde{W} \oplus K^0) = L(\Delta Y) \tag{3.2}$$

В среднем для каждого такого уравнения будет существовать одно решение. Все ложные ключи встретятся один раз, искомый ключ встретится два раза.

Оценка трудоёмкости атаки

Параметр	Оценка
Связанные ключи	2
8-байтовых блоков памяти(K^0)	$2^{22} = p^{-1} \cdot 2 \cdot 255 \cdot 2^{-1}$
Операций доступа к памяти	$2^{25} = p^{-1} \cdot 2 \cdot 255 \cdot 8$

Таблица 2 – Трудоёмкость атаки.

3.3 Описание программной реализации

Реализация алгоритмов зашифрования и расшифрования

Раундовая функция шифра Khazad имеет вид:

$$y = X[K^i] \circ L \circ S(x) \tag{3.3}$$

В спецификации шифра [1] авторы сами предлагают способ эффективной реализации композиции преобразований $L \circ S(x)$. Если обратить внимание на то, каким образом матрица 1.2 производит умножение строки из восьми байт в конечном поле, то можно заметить, что по сути каждый i-й байт умножается на i-й столбец матрицы в конечном поле. Только в нашем случае производится умножение не самих байтов строки, а отображенных под действием S-преобразования. Будем пользоваться восемью предпосчитанными таблицами

$T_i[x] = S[x] \cdot H[i], i \in \overline{0,7}$, где $H[i]$ обозначает i-ю строку матрицы H. Выглядеть они будут так:

$$
\begin{aligned}
T_0[x] &= S[x] \cdot [1,3,4,5,6,8,B,7],\\
T_1[x] &= S[x] \cdot [3,1,5,4,8,6,7,B],\\
T_2[x] &= S[x] \cdot [4,5,1,3,B,7,6,8],\\
T_3[x] &= S[x] \cdot [5,4,3,1,7,B,8,6],\\
T_4[x] &= S[x] \cdot [6,8,B,7,1,3,4,5],\\
T_5[x] &= S[x] \cdot [8,6,7,B,3,1,5,4],\\
T_6[x] &= S[x] \cdot [B,7,6,8,4,5,1,3],\\
T_7[x] &= S[x] \cdot [7,B,8,6,5,4,3,1].
\end{aligned}
$$

В итоге для $X, Y \in GF(2^8)^8$ имеем

$$Y = L \circ S(X) = \bigoplus_{i=0}^{7} T_i[X[i]].$$

Таблицы T приведены в приложении в листинге 5.

Шифраторы 4-х рауандового Khazad'a с изменённой ключевой развёрткой реализованы в виде класса khazad_reduced, имеющего следующий API:

- uint64_t encrypt(uint64_t plaintext) - метод для зашифрования
- uint64_t decrypt(uint64_t ciphertext) - метод для расшифрования

Они реализованы с помощью таблиц T. Также для нужд отладки поддержан API со стандартными преобразования шифра в качестве статических методов:

- uint64_t L(uint64_t s) - метод, реализующий линейное преобразование
- uint64_t S(uint64_t state) - метод, реализующий нелинейное преобразование

Атака реализована в виде класса Experiment, определение и реализация которого приведены в листингах 3 и 4. Рассмотрим подробнее реализацию методов.

Метод, решающий уравнение 3.2 относительно раундового ключа K^0, имеет сигнатуру:

```
auto
RKB4RExp::brute_key(uint64_t z1, uint64_t z2, uint64_t ltheta)
-> std::optional<std::vector<std::vector<uint8_t>>>;
```

Данный метод возвращает класс монады *std::optional*, содержащий либо множество ключей K^0 в виде *std::vector< std::vector<uint8_t> >*, либо пустое значение *std::nullopt*, если такое множество построить нельзя.

ГЛАВА 4

Теоретическая атака методом бумеранга в модели связанных ключей на 5 раундов шифра Khazad с модифицированной ключевой развёрткой

Существует атака на 5 раундов шифра Khazad [19], имеющая следующую трудоёмкость: 2^{64+28} по времени и $\simeq 2^{64}$ по данным. Однако, такая оценка имеет место при полном переборе раундового ключа K^0. Если же раундовый ключ K^0 скомпрометирован, то сложность атаки уменьшается до всего лишь 256 избранных открытых текстов и 2^{28} обращений к S-box'у. Автор статьи [19] пишет, что найти способ нахождения ключа K^0 для шифра с оригинальной функцией ключевой развёртки эффективнее, чем полный перебор - не удалось. В предыдущем разделе мы реализовали атаку на четыре раунда, с помощью которой можно эффективно найти раундовый ключ K^0. Это как раз то, что нам нужно, поэтому попробуем развить атаку на ещё один дополнительный раунд. Так как атака будет оптимизировать полный перебор, при подсчёте трудоёмкости будем пользоваться оценками сверху, чтобы иметь представление насколько хорошим может быть потенциальное улучшение.

При построении данной атаки будем пользоваться техниками, использованными при построении атаки на четырехраундовое шифрпреобразование.

Разобьём процесс шифрования на три части:

$$\begin{gathered} E(P) = E_2 \circ S \circ E_1(P), \\ E(P) = X[K^5]SX[K^4]L \circ S \circ X[K^3]LSX[K^2]LSX[K^1]LSX[K^0](P) \\ E_1 = X[K^3]LSX[K^2]LSX[K^1]LSX[K^0] \\ E_2 = X[K^5]SX[K^4]L \end{gathered} \tag{4.1}$$

4.1 Построение дифференциального пути внутри сети Фейстеля ключевой развёртки

Выбираем разность исходных раундовых ключей таким же образом, как и в предыдущей атаке:

$$\begin{aligned} \Delta K_0 &= 0 \\ \Delta K_1 &= \kappa \end{aligned} \tag{4.2}$$

Получаем следующий дифференциальный путь внутри функции ключевой развёртки:

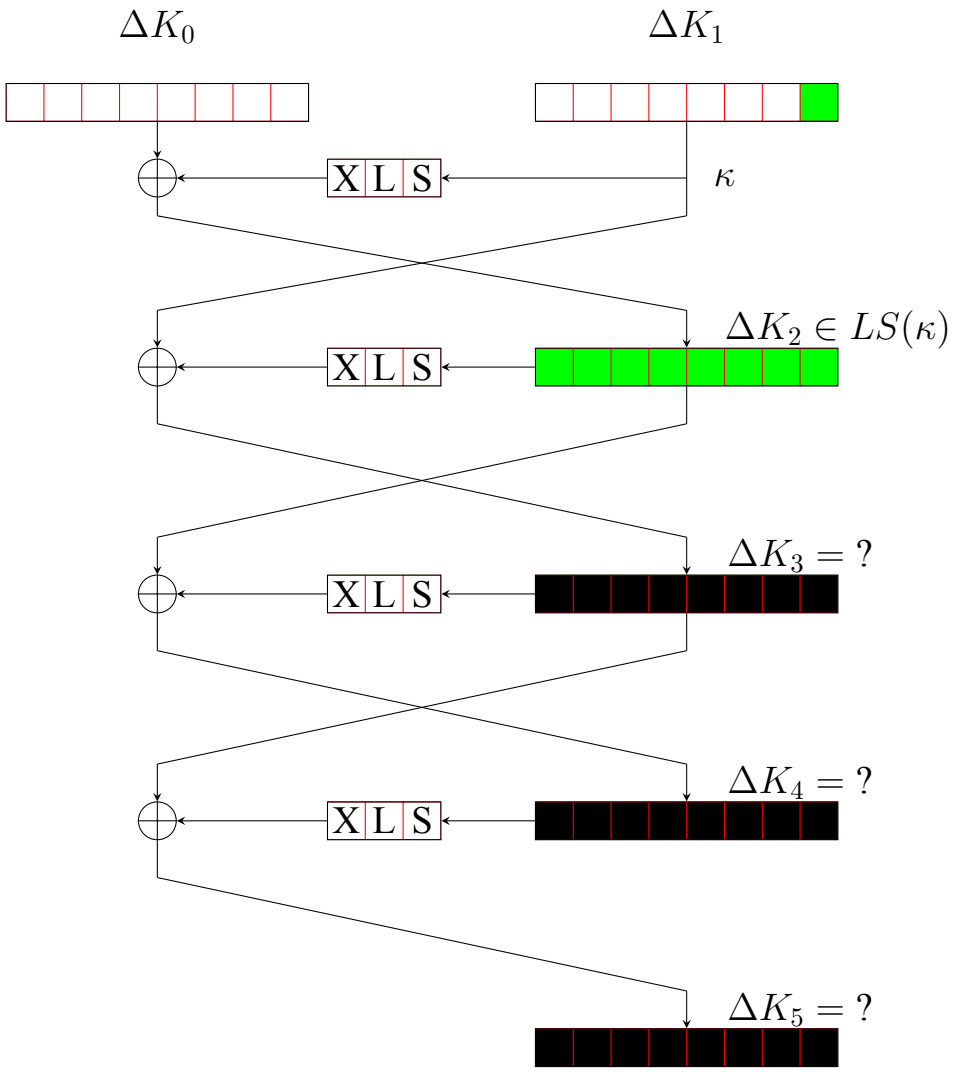

Рисунок 10 – Дифференциальный путь внутри сети Фейстеля модифицированной ключевой развертки для атаки методом бумеранга на связанных ключах на 5 раундов шифра Khazad.

В итоге имеем следующий набор разностей:

$$\begin{aligned} \Delta K_0 &= 0 \\ \Delta K_1 &= \kappa \\ \Delta K_2 &= LS(\kappa) \\ \Delta K_3 &= ? \\ \Delta K_4 &= ? \\ \Delta K_5 &= ? \end{aligned} \tag{4.3}$$

4.2 Построение дифференциального пути на связанных ключах внутри шифрпреобразования

На вход шифраторов будем подавать один и тот же открытый текст, сгенерированный случайным образом $E_K(P) = Q$ и $E_{\tilde{K}}(P) = \tilde{Q}$. На выходе имеем пары шифротекстов, точную разность которых мы предугадать не можем.

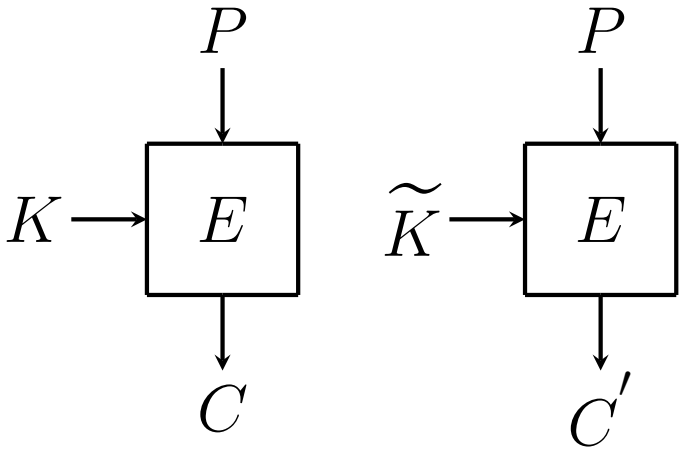

Рисунок 11 – Модель атаки

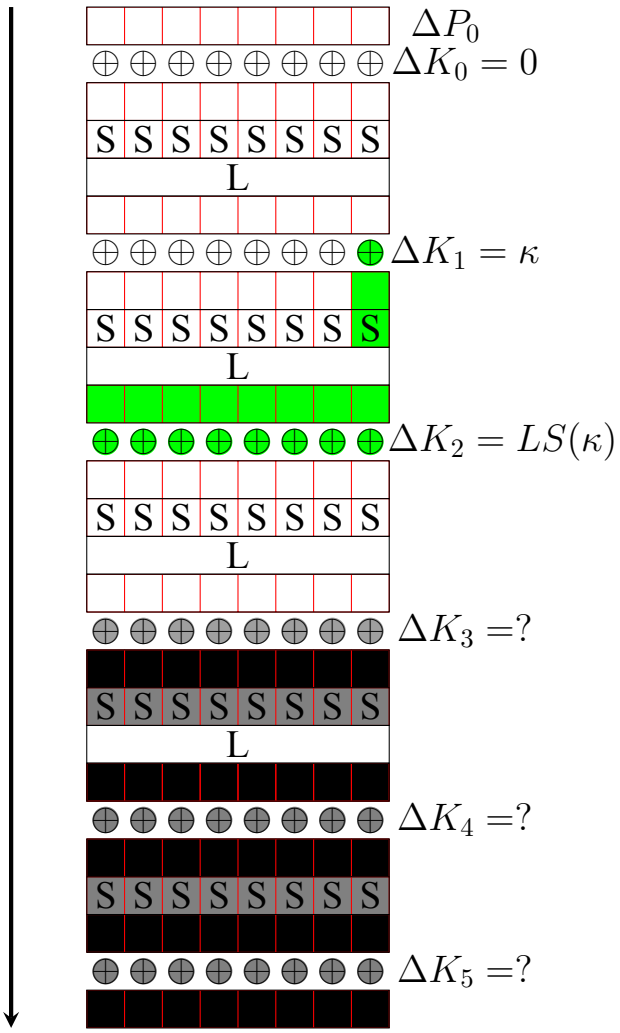

Рисунок 12 – Дифференциальный путь на связанных ключах внутри шифрпреобразования в направлении ”сверху вниз” для атаки на 5 раундов.

Стоит обратить внимание на наличие возможной локальной коллизии после наложения ключей K_2. Её вероятность оценивается как:

$$\begin{gathered} Pr(X[K^2]LSX[K^1]LSX[K^0](P) \oplus \\ X[\widetilde{K^2}]LSX[\widetilde{K^1}]LSX[K^0](P) = 0) \geq 2^{-5} \end{gathered} \tag{4.4}$$

К каждому из полученных шифротекстов прибавляем разность δ отличающуюся только в одном байте $Q \oplus \delta = Z$ и $\tilde{Q} \oplus \delta = \tilde{Z}$ и идём вверх на несвязанных ключах.

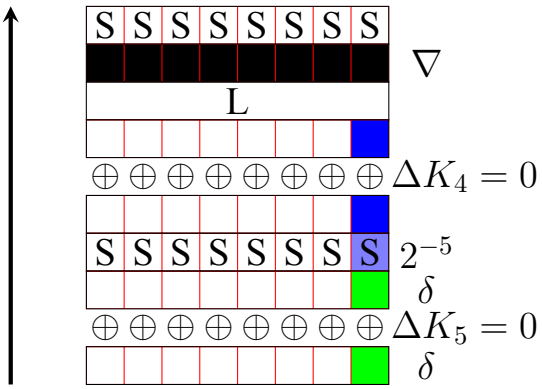

Рисунок 13 – Дифференциальный путь на несвязанных ключах внутри шифр-преобразования в направлении ”снизу вверх” для атаки на 5 раундов.

$$\begin{gathered}\nabla = LX[K^4]SX[K^5](Z) \oplus LX[K^4]SX[K^5](Q),\\ \tilde{\nabla} = LX[K^4]SX[K^5](\tilde{Z}) \oplus LX[K^4]SX[K^5](\tilde{Q}),\\ Pr(\nabla = \tilde{\nabla}) \leq 2^{-5}.\end{gathered} \tag{4.5}$$

Далее на пути атаки стоит S-box switch.

$$\begin{gathered}D = X[K^3]LSX[K^2]LSX[K^1]LSX[K^0](P),\\ \Delta = \Delta K_3,\\ Pr(S(S(D) \oplus \nabla) \oplus S(S(D \oplus \Delta) \oplus \nabla) = \Delta) \leq 2^{-40}.\end{gathered} \tag{4.6}$$

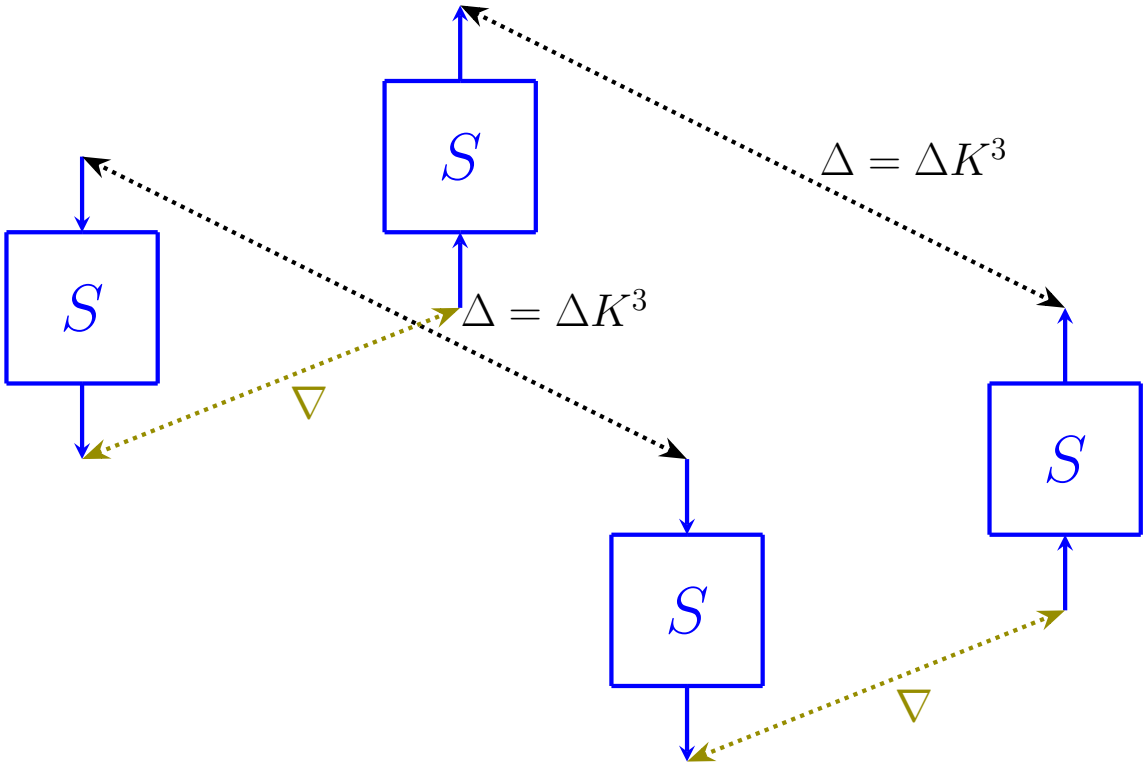

Рисунок 14 – Sbox соединение

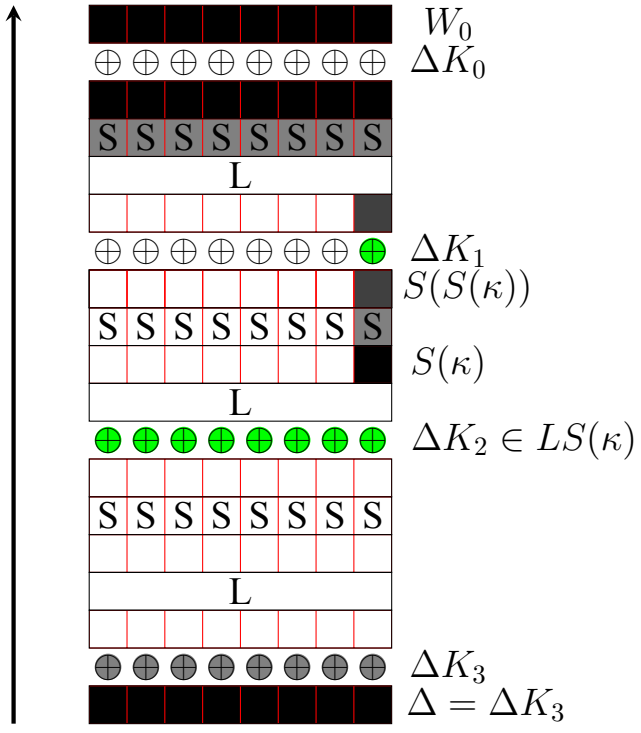

Рисунок 15 – Дифференциальный путь на связанных ключах внутри шифрпреобразования в направлении ”снизу вверх”.

Далее схема атаки совпадает с атакой на 4 раунда.

4.3 Оценка трудоёмкости атаки

Как итог получили заметное ухудшение вероятности с оценкой в лучшем случае 2^{-50}.

Параметр	Оценка
Связанные ключи	2
8-байтовых блоков памяти(K^0)	$2^{58} = p^{-1} \cdot 2 \cdot 255 \cdot 2^{-1}$
Операций доступа к памяти	$2^{61} = p^{-1} \cdot 2 \cdot 255 \cdot 8$

Таблица 3 – Трудоёмкость атаки.

На персональном компьютере такую атаку не реализовать. Однако стоит понимать, что, найдя таким образом ключ K_0, можно исключить необходимость полного перебора ключей в атаке [19], сложность которой пренебрежимо мала по сравнению со сложностью предложенной атаки методом бумеранга по восстановлению ключа, следовательно обе части мастер ключа можно восстановить примерно со сложностью порядка сложности представленной атаки. Такая комбинация атак даёт суммарную сложность по времени $2^{61} + 2^{28} \approx 2^{61}$ вместо 2^{64+28} и по данным $2^{58} + 2^8 \approx 2^{58}$ вместо полного словаря размером 2^{64}, что является серьёзным улучшением.

Заключение

В результате данной работы были предложены две атаки в модели связанных ключей на шифр Khazad, были представлены оценки трудоёмкости реализации данных атак. Для атаки на четыре раунда была представлена программная реализация, подтверждающая корректность алгоритма и оценок трудоёмкости. Теоретическая атака на пять раундов позволяет получить существенный выигрыш в оценках по времени и памяти при использовании её в качестве дополнения к атаке [19].

При построении эффективных атак желательно уметь строить оптимальные дифференциальные пути на связанных ключах, имеющие как можно меньше активных байтов для улучшения вероятностей дифференциальных путей. Отдельным перспективным направлением для исследования является разработка и программная реализация алгоритмов и эвристик, которые позволяли бы для данного блочного шифра найти самые лучшие дифференциальные пути в модели связанных ключей. Подобные инструменты позволили бы проводить криптоанализ в рассмотренной модели более эффективно.

Литература

[1] Paulo Barreto and Vincent Rijmen. The khazad legacy-level block cipher. 01 2000.

[2] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 491–506, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[3] Eli Biham, Orr Dunkelman, and Nathan Keller. Related-key boomerang and rectangle attacks. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 507–525, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[4] Eli Biham, Orr Dunkelman, and Nathan Keller. A unified approach to related-key attacks. In Kaisa Nyberg, editor, *Fast Software Encryption*, pages 73–96, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[5] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology-CRYPTO' 90*, pages 2–21, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[6] Alex Biryukov. Related key attack. In Henk C. A. van Tilborg, editor, *Encyclopedia of Cryptography and Security*, pages 518–519, Boston, MA, 2005. Springer US.

[7] Alex Biryukov and Dmitry Khovratovich. Related-key cryptanalysis of the full aes-192 and aes-256. pages 1–18, 2009.

[8] Alex Biryukov, Dmitry Khovratovich, and Léo Perrin. Multiset-algebraic cryptanalysis of reduced kuznyechik, khazad, and secret spns. *IACR Transactions on Symmetric Cryptology*, 2016(2):226–247, Feb. 2017.

[9] Alex Biryukov and Ivica Nikolić. Automatic search for related-key differential characteristics in byte-oriented block ciphers: Application to aes, camellia, khazad and others. 2010. https://eprint.iacr.org/2010/248.

[10] Céline Blondeau and Benoît Gérard. Multiple differential cryptanalysis: Theory and practice. In Antoine Joux, editor, *Fast Software Encryption*, pages 35–54, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[11] B. A. Pogorelov D. A. Burov 1. An attack on 6 rounds of khazad. *МАТЕМАТИЧЕСКИЕ ВОПРОСЫ КРИПТОГРАФИИ*, 2016(2):35–46, 2016.

[12] Liam Keliher and Jiayuan Sui. Exact maximum expected differential and linear probability for 2-round advanced encryption standard (aes). Cryptology ePrint Archive, Paper 2005/321, 2005. https://eprint.iacr.org/2005/321.

[13] John Kelsey, Bruce Schneier, and David Wagner. Related-key cryptanalysis of 3-way, biham-des,cast, des-x, newdes, rc2, and tea. In Yongfei Han, Tatsuaki Okamoto, and Sihan Qing, editors, *Information and Communications Security*, pages 233–246, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[14] Vitaly Kiryukhin. Exact maximum expected differential and linear probability for 2-round kuznyechik. Cryptology ePrint Archive, Paper 2018/1085, 2018. https://eprint.iacr.org/2018/1085.

[15] Vitaly Kiryukhin. Related-key attack on 5-round kuznyechik. 2019. https://eprint.iacr.org/2019/1249.

[16] Lars R. Knudsen. Truncated and higher order differentials. In Bart Preneel, editor, *Fast Software Encryption*, pages 196–211, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[17] Lars Ramkilde Knudsen. Cryptanalysis of loki. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, pages 22–35, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[18] Xuejia Lai, James L. Massey, and Sean Murphy. Markov ciphers and differential cryptanalysis. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, pages 17–38, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[19] Frédéric Muller. A new attack against khazad. In *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, volume 2894 of *Lecture Notes in Computer Science*, pages 347–358. Springer, 2003.

[20] Claude E. Shannon. Communication theory of secrecy systems. *Bell Syst. Tech. J.*, 28(4):656–715, 1949.

[21] David Wagner. The boomerang attack. In Lars Knudsen, editor, *Fast Software Encryption*, pages 156–170, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[22] Meiqin Wang. Differential cryptanalysis of reduced-round present. In Serge Vaudenay, editor, *Progress in Cryptology – AFRICACRYPT 2008*, pages 40–49, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[23] В. А. Кирюхин. Атака методом бумеранга на связанных ключах на 5 раундов шифра Кузнечик. *Обозрение прикладной промышленной математики*, 26(3), 2019.

Приложение

Листинг 1 – randomizer.hpp

```
//
// Created by kadius on 27.12.23.
//

#ifndef CRYPTOGRAPHY_RANDOMIZER_HPP
#define CRYPTOGRAPHY_RANDOMIZER_HPP

#include <random>


struct Randomizer {
        uint8_t rnd();
        uint8_t rnd_notnull();
        uint64_t rnd64();
        private:
        std::random_device seed;
        std::mt19937_64 rng{seed()};
        std::uniform_int_distribution<uint8_t> dst{0, 0xff};
        std::uniform_int_distribution<uint8_t> dst_notnull{1, 0xff};
};

#endif //CRYPTOGRAPHY_RANDOMIZER_HPP
```

Листинг 2 – randomizer.cpp

```
#include "randomizer.hpp"

uint8_t Randomizer::rnd() {
        return dst(rng);
}

uint8_t Randomizer::rnd_notnull() {
        return dst_notnull(rng);
}

uint64_t Randomizer::rnd64() {
        uint64_t res = 0;
        for(int i{}; i < 8; ++i) {
                res <<= 8;
                res ^= rnd();
        }

        return res;
```

```
}
```

Листинг 3 – Experiment.hpp

```
//
// Created by kadius on 04.12.23.
//

#ifndef CRYPTOGRAPHY_EXPERIMENT_HPP
#define CRYPTOGRAPHY_EXPERIMENT_HPP

#include <KHAZAD.hpp>
#include <randomizer.hpp>
#include <unordered_set>
#include <optional>
#include <iostream>
#include <iomanip>

struct RKB4RExp {
    RKB4RExp(khazad_reduced& enc1,
             khazad_reduced& enc2):encryptor1(enc1), encryptor2(enc2) {}

    //randomly initializes keys
    RKB4RExp();

    void Run();

    private:
    void reserve(std::unordered_set<uint64_t>& set) {
        set.reserve(2<<12);
    }

    template<size_t N>
    uint64_t arr_to_64(const uint8_t(&arr)[N]);

    auto
    brute_key(uint64_t z1, uint64_t z2, uint64_t ltheta)
    -> std::optional<std::vector<std::vector<uint8_t>>>;

    bool variants(const std::vector<std::vector<uint8_t>>& v,
                  std::unordered_set<uint64_t>& map);

    private:
    Randomizer randomizer;
    khazad_reduced encryptor1;
    khazad_reduced encryptor2;
```

```
    uint64_t plaintext{};
};

#endif //CRYPTOGRAPHY_EXPERIMENT_HPP
```

Листинг 4 – Experiment.cpp

```
#include "Experiment.hpp"
#include "randomizer.hpp"

RKB4RExp::RKB4RExp(){
    uint64_t key1 = randomizer.rnd64();
    std::cout << "TRY RECOVER KEY1 = " << std::hex << key1 << std::endl;
    uint64_t key2 = randomizer.rnd64();

    encryptor1 = khazad_reduced(key1, key2);
    encryptor2 = khazad_reduced(key1, key2^(randomizer.rnd_notnull()));
    std::cout << "when KEY2 = " << std::hex <<
    key2 << " and Key2' = " << encryptor2.ekeys_[1] << std::endl;
}

void RKB4RExp::Run()  {
    std::unordered_set<uint64_t> set;
    reserve(set);

    for (int t{}; /*t < (2 << 16)*/; ++t) {
        plaintext = randomizer.rnd64();

        uint64_t q1 = encryptor1.encrypt(plaintext);
        uint64_t q2 = encryptor2.encrypt(plaintext);

        uint64_t diff = randomizer.rnd_notnull();

        uint64_t z1 = q1^diff;
        uint64_t z2 = q2^diff;

        uint64_t w1 = encryptor1.decrypt(z1);
        uint64_t w2 = encryptor2.decrypt(z2);

        if(w1 != w2)
        for (int theta{1}; theta < 256; ++theta) {
            auto v = brute_key(w1, w2, khazad_reduced::L(theta));
            if (v != std::nullopt)
            if (variants(*v, set))
            goto ex;
        } //for
    }
```

```
    ex:
    if(true){}
}

auto
RKB4RExp::brute_key(uint64_t z1, uint64_t z2, uint64_t ltheta)
-> std::optional<std::vector<std::vector<uint8_t>>>
{
    //    std::cout << "brute key called" << std::endl;
    uint64_t mask = 0x00000000000000FF;

    uint8_t tmp1[8];
    tmp1[0] = (uint8_t)(z1 & mask);
    tmp1[1] = (uint8_t)((z1 >> 8) & mask);
    tmp1[2] = (uint8_t)((z1 >> 16) & mask);
    tmp1[3] = (uint8_t)((z1 >> 24) & mask);
    tmp1[4] = (uint8_t)((z1 >> 32) & mask);
    tmp1[5] = (uint8_t)((z1 >> 40) & mask);
    tmp1[6] = (uint8_t)((z1 >> 48) & mask);
    tmp1[7] = (uint8_t)((z1 >> 56) & mask);

    uint8_t tmp2[8];
    tmp2[0] = (uint8_t)(z2 & mask);
    tmp2[1] = (uint8_t)((z2 >> 8) & mask);
    tmp2[2] = (uint8_t)((z2 >> 16) & mask);
    tmp2[3] = (uint8_t)((z2 >> 24) & mask);
    tmp2[4] = (uint8_t)((z2 >> 32) & mask);
    tmp2[5] = (uint8_t)((z2 >> 40) & mask);
    tmp2[6] = (uint8_t)((z2 >> 48) & mask);
    tmp2[7] = (uint8_t)((z2 >> 56) & mask);

    uint8_t ltheta_tmp[8];
    ltheta_tmp[0] = (uint8_t)(ltheta & mask);
    ltheta_tmp[1] = (uint8_t)((ltheta >> 8) & mask);
    ltheta_tmp[2] = (uint8_t)((ltheta >> 16) & mask);
    ltheta_tmp[3] = (uint8_t)((ltheta >> 24) & mask);
    ltheta_tmp[4] = (uint8_t)((ltheta >> 32) & mask);
    ltheta_tmp[5] = (uint8_t)((ltheta >> 40) & mask);
    ltheta_tmp[6] = (uint8_t)((ltheta >> 48) & mask);
    ltheta_tmp[7] = (uint8_t)((ltheta >> 56) & mask);

    std::vector<std::vector<uint8_t>> variants(8, std::vector<uint8_t>());
    for(int i{}; i < 8; ++i) {
        for (int x{}; x < 256; ++x)
        if ((khazad_reduced::S(tmp1[i] ^ x) ^
```

```
            khazad_reduced::S(tmp2[i] ^ x)) == ltheta_tmp[i])
        variants.at(i).push_back(x);

        if(variants.at(i).empty()) return std::nullopt;
    }

    return variants;
}

bool RKB4RExp::variants(const std::vector<std::vector<uint8_t>>& v,
                        std::unordered_set<uint64_t>& map) {
        auto it0 = v.at(0).begin();
        auto it1 = v.at(1).begin();
        auto it2 = v.at(2).begin();
        auto it3 = v.at(3).begin();
        auto it4 = v.at(4).begin();
        auto it5 = v.at(5).begin();
        auto it6 = v.at(6).begin();
        auto it7 = v.at(7).begin();

        auto eit0 = v.at(0).end();
        auto eit1 = v.at(1).end();
        auto eit2 = v.at(2).end();
        auto eit3 = v.at(3).end();
        auto eit4 = v.at(4).end();
        auto eit5 = v.at(5).end();
        auto eit6 = v.at(6).end();
        auto eit7 = v.at(7).end();

        uint8_t tmp[8] = {0};

      for(; it0 != eit0; ++it0) {
        tmp[0] = *it0;

        for(; it1 != eit1; ++it1) {
          tmp[1] = *it1;

          for(; it2 != eit2; ++it2) {
                        tmp[2] = *it2;

            for(; it3 != eit3; ++it3) {
              tmp[3] = *it3;

              for(; it4 != eit4; ++it4) {
                tmp[4] = *it4;
```

```
            for(; it5 != eit5; ++it5) {
              tmp[5] = *it5;

              for(; it6 != eit6; ++it6) {
                tmp[6] = *it6;

                for(; it7 != eit7; ++it7) {
                tmp[7] = *it7;
                uint64_t key = arr_to_64(tmp);
              if(map.count(key) == 0) {
                map.insert(key);
              } else {
                std::cout << "WTF" << std::endl;
                std::cout << std::setw(16)
                << std::setfill('0')
                << std::hex << key <<
                "<-KEY HAS BEEN FOUND" <<
                std::endl;
                return true;
                                }
              }
              it7 = v.at(7).begin();
            }
            it6 = v.at(6).begin();
            it7 = v.at(7).begin();
                      }
          it5 = v.at(5).begin();
          it6 = v.at(6).begin();
          it7 = v.at(7).begin();
        }
        it4 = v.at(4).begin();
        it5 = v.at(5).begin();
        it6 = v.at(6).begin();
        it7 = v.at(7).begin();
      }
      it3 = v.at(3).begin();
      it4 = v.at(4).begin();
      it5 = v.at(5).begin();
      it6 = v.at(6).begin();
      it7 = v.at(7).begin();
    }
    it2 = v.at(2).begin();
    it3 = v.at(3).begin();
    it4 = v.at(4).begin();
    it5 = v.at(5).begin();
    it6 = v.at(6).begin();
```

```
      it7 = v.at(7).begin();
    }
    it1 = v.at(1).begin();
    it2 = v.at(2).begin();
    it3 = v.at(3).begin();
    it4 = v.at(4).begin();
    it5 = v.at(5).begin();
    it6 = v.at(6).begin();
    it7 = v.at(7).begin();
  }
  return false;
}

template<size_t N>
uint64_t RKB4RExp::arr_to_64(const uint8_t (&arr)[N]) {
        uint64_t x = 0;

        x <<= 8;
        x ^= arr[7];
        x <<= 8;
        x ^= arr[6];
        x <<= 8;
        x ^= arr[5];
        x <<= 8;
        x ^= arr[4];
        x <<= 8;
        x ^= arr[3];
        x <<= 8;
        x ^= arr[2];
        x <<= 8;
        x ^= arr[1];
        x <<= 8;
        x ^= arr[0];

        return x;
}
```

Листинг 5 – TTables.hpp

```
        const block T0[256] = {
                0xbad3d268bbb96a01, 0x54fc4d19e59a66b1,
                0x2f71bc93e26514cd, 0x749ccdb925871b51,
                0x53f55102f7a257a4, 0xd3686bb8d0d6be03,
                0xd26b6fbdd6deb504, 0x4dd72964b35285fe,
                0x50f05d0dfdba4aad, 0xace98a26cf09e063,
                0x8d8a0e83091c9684, 0xbfdcc679a5914d1a,
                0x7090ddad3da7374d, 0x52f65507f1aa5ca3,
```

```
0x9ab352c87ba417e1 , 0x4cd42d61b55a8ef9 ,
0xea238f65460320ac , 0xd56273a6c4e68411 ,
0x97a466f155cc68c2 , 0xd16e63b2dcc6a80d ,
0x3355ccffaa85d099 , 0x51f35908fbb241aa ,
0x5bed712ac7e20f9c , 0xa6f7a204f359ae55 ,
0xde7f5f81febec120 , 0x48d83d75ad7aa2e5 ,
0xa8e59a32d729cc7f , 0x99b65ec771bc0ae8 ,
0xdb704b90e096e63b , 0x3256c8faac8ddb9e ,
0xb7c4e65195d11522 , 0xfc19d72b32b3aace ,
0xe338ab48704b7393 , 0x9ebf42dc63843bfd ,
0x91ae7eef41fc52d0 , 0x9bb056cd7dac1ce6 ,
0xe23baf4d76437894 , 0xbbd0d66dbdb16106 ,
0x41c319589b32f1da , 0x6eb2a5cb7957e517 ,
0xa5f2ae0bf941b35c , 0xcb400bc08016564b ,
0x6bbdb1da677fc20c , 0x95a26efb59dc7ecc ,
0xa1febe1fe1619f40 , 0xf308eb1810cbc3e3 ,
0xb1cefe4f81e12f30 , 0x0206080a0c10160e ,
0xcc4917db922e675e , 0xc45137f3a26e3f66 ,
0x1d2774694ee8cf53 , 0x143c504478a09c6c ,
0xc3582be8b0560e73 , 0x63a591f2573f9a34 ,
0xda734f95e69eed3c , 0x5de76934d3d2358e ,
0x5fe1613edfc22380 , 0xdc79578bf2aed72e ,
0x7d87e99413cf486e , 0xcd4a13de94266c59 ,
0x7f81e19e1fdf5e60 , 0x5aee752fc1ea049b ,
0x6cb4adc17547f319 , 0x5ce46d31d5da3e89 ,
0xf704fb0c08ebefff , 0x266a98bed42d47f2 ,
0xff1cdb2438abb7c7 , 0xed2a937e543b11b9 ,
0xe825876f4a1336a2 , 0x9dba4ed3699c26f4 ,
0x6fb1a1ce7f5fee10 , 0x8e8f028c03048b8d ,
0x192b647d56c8e34f , 0xa0fdba1ae7699447 ,
0xf00de7171ad3deea , 0x89861e97113cba98 ,
0x0f113c332278692d , 0x07091c1b12383115 ,
0xafec8629c511fd6a , 0xfb10cb30208b9bdb ,
0x0818202830405838 , 0x153f54417ea8976b ,
0x0d1734392e687f23 , 0x040c101418202c1c ,
0x0103040506080b07 , 0x64ac8de94507ab21 ,
0xdf7c5b84f8b6ca27 , 0x769ac5b329970d5f ,
0x798bf9800bef6472 , 0xdd7a538ef4a6dc29 ,
0x3d47f4c98ef5b2b3 , 0x163a584e74b08a62 ,
0x3f41fcc382e5a4bd , 0x3759dcebb2a5fc85 ,
0x6db7a9c4734ff81e , 0x3848e0d890dd95a8 ,
0xb9d6de67b1a17708 , 0x7395d1a237bf2a44 ,
0xe926836a4c1b3da5 , 0x355fd4e1beb5ea8b ,
0x55ff491ce3926db6 , 0x7193d9a83baf3c4a ,
0x7b8df18a07ff727c , 0x8c890a860f149d83 ,
0x7296d5a731b72143 , 0x88851a921734b19f ,
```

```
0xf607ff090ee3e4f8 , 0x2a7ea882fc4d33d6 ,
0x3e42f8c684edafba , 0x5ee2653bd9ca2887 ,
0x27699cbbd2254cf5 , 0x46ca0543890ac0cf ,
0x0c14303c28607424 , 0x65af89ec430fa026 ,
0x68b8bdd56d67df05 , 0x61a399f85b2f8c3a ,
0x03050c0f0a181d09 , 0xc15e23e2bc46187d ,
0x57f94116ef827bb8 , 0xd6677fa9cefe9918 ,
0xd976439aec86f035 , 0x58e87d25cdfa1295 ,
0xd875479fea8efb32 , 0x66aa85e34917bd2f ,
0xd7647bacc8f6921f , 0x3a4ee8d29ccd83a6 ,
0xc84507cf8a0e4b42 , 0x3c44f0cc88fdb9b4 ,
0xfa13cf35268390dc , 0x96a762f453c463c5 ,
0xa7f4a601f551a552 , 0x98b55ac277b401ef ,
0xec29977b52331abe , 0xb8d5da62b7a97c0f ,
0xc7543bfca876226f , 0xaeef822cc319f66d ,
0x69bbb9d06b6fd402 , 0x4bdd317aa762bfec ,
0xabe0963ddd31d176 , 0xa9e69e37d121c778 ,
0x67a981e64f1fb628 , 0x0a1e28223c504e36 ,
0x47c901468f02cbc8 , 0xf20bef1d16c3c8e4 ,
0xb5c2ee5b99c1032c , 0x226688aacc0d6bee ,
0xe532b356647b4981 , 0xee2f9f715e230cb0 ,
0xbedfc27ca399461d , 0x2b7dac87fa4538d1 ,
0x819e3ebf217ce2a0 , 0x1236485a6c90a67e ,
0x839836b52d6cf4ae , 0x1b2d6c775ad8f541 ,
0x0e1238362470622a , 0x23658cafca0560e9 ,
0xf502f30604fbf9f1 , 0x45cf094c8312ddc6 ,
0x216384a5c61576e7 , 0xce4f1fd19e3e7150 ,
0x49db3970ab72a9e2 , 0x2c74b09ce87d09c4 ,
0xf916c33a2c9b8dd5 , 0xe637bf596e635488 ,
0xb6c7e25493d91e25 , 0x2878a088f05d25d8 ,
0x17395c4b72b88165 , 0x829b32b02b64ffa9 ,
0x1a2e68725cd0fe46 , 0x8b80169d1d2cac96 ,
0xfe1fdf213ea3bcc0 , 0x8a8312981b24a791 ,
0x091b242d3648533f , 0xc94603ca8c064045 ,
0x879426a1354cd8b2 , 0x4ed2256bb94a98f7 ,
0xe13ea3427c5b659d , 0x2e72b896e46d1fca ,
0xe431b75362734286 , 0xe03da7477a536e9a ,
0xeb208b60400b2bab , 0x90ad7aea47f459d7 ,
0xa4f1aa0eff49b85b , 0x1e22786644f0d25a ,
0x85922eab395ccebc , 0x60a09dfd5d27873d ,
0x0000000000000000 , 0x256f94b1de355afb ,
0xf401f70302f3f2f6 , 0xf10ee3121cdbd5ed ,
0x94a16afe5fd475cb , 0x0b1d2c273a584531 ,
0xe734bb5c686b5f8f , 0x759fc9bc238f1056 ,
0xef2c9b74582b07b7 , 0x345cd0e4b8bde18c ,
0x3153c4f5a695c697 , 0xd46177a3c2ee8f16 ,
```

```
        0xd06d67b7dacea30a, 0x869722a43344d3b5,
        0x7e82e59b19d75567, 0xadea8e23c901eb64,
        0xfd1ad32e34bba1c9, 0x297ba48df6552edf,
        0x3050c0f0a09dcd90, 0x3b4decd79ac588a1,
        0x9fbc46d9658c30fa, 0xf815c73f2a9386d2,
        0xc6573ff9ae7e2968, 0x13354c5f6a98ad79,
        0x060a181e14303a12, 0x050f14111e28271b,
        0xc55233f6a4663461, 0x113344556688bb77,
        0x7799c1b62f9f0658, 0x7c84ed9115c74369,
        0x7a8ef58f01f7797b, 0x7888fd850de76f75,
        0x365ad8eeb4adf782, 0x1c24706c48e0c454,
        0x394be4dd96d59eaf, 0x59eb7920cbf21992,
        0x1828607850c0e848, 0x56fa4513e98a70bf,
        0xb3c8f6458df1393e, 0xb0cdfa4a87e92437,
        0x246c90b4d83d51fc, 0x206080a0c01d7de0,
        0xb2cbf2408bf93239, 0x92ab72e04be44fd9,
        0xa3f8b615ed71894e, 0xc05d27e7ba4e137a,
        0x44cc0d49851ad6c1, 0x62a695f751379133,
        0x103040506080b070, 0xb4c1ea5e9fc9082b,
        0x84912aae3f54c5bb, 0x43c511529722e7d4,
        0x93a876e54dec44de, 0xc25b2fedb65e0574,
        0x4ade357fa16ab4eb, 0xbddace73a9815b14,
        0x8f8c0689050c808a, 0x2d77b499ee7502c3,
        0xbcd9ca76af895013, 0x9cb94ad66f942df3,
        0x6abeb5df6177c90b, 0x40c01d5d9d3afadd,
        0xcf4c1bd498367a57, 0xa2fbb210eb798249,
        0x809d3aba2774e9a7, 0x4fd1216ebf4293f0,
        0x1f217c6342f8d95d, 0xca430fc5861e5d4c,
        0xaae39238db39da71, 0x42c61557912aecd3,
};

const block T1[256] = {
        0xd3ba68d2b9bb016a, 0xfc54194d9ae5b166,
        0x712f93bc65e2cd14, 0x9c74b9cd8725511b,
        0xf5530251a2f7a457, 0x68d3b86bd6d003be,
        0x6bd2bd6fded604b5, 0xd74d642952b3fe85,
        0xf0500d5dbafdad4a, 0xe9ac268a09cf63e0,
        0x8a8d830e1c098496, 0xdcbf79c691a51a4d,
        0x9070addda73d4d37, 0xf6520755aaf1a35c,
        0xb39ac852a47be117, 0xd44c612d5ab5f98e,
        0x23ea658f0346ac20, 0x62d5a673e6c41184,
        0xa497f166cc55c268, 0x6ed1b263c6dc0da8,
        0x5533ffcc85aa99d0, 0xf3510859b2fbaa41,
        0xed5b2a71e2c79c0f, 0xf7a604a259f355ae,
        0x7fde815fbefe20c1, 0xd848753d7aade5a2,
        0xe5a8329a29d77fcc, 0xb699c75ebc71e80a,
```

```
0x70db904b96e03be6 , 0x5632fac88dac9edb ,
0xc4b751e6d1952215 , 0x19fc2bd7b332ceaa ,
0x38e348ab4b709373 , 0xbf9edc428463fd3b ,
0xae91ef7efc41d052 , 0xb09bcd56ac7de61c ,
0x3be24daf43769478 , 0xd0bb6dd6b1bd0661 ,
0xc3415819329bdaf1 , 0xb26ecba5577917e5 ,
0xf2a50bae41f95cb3 , 0x40cbc00b16804b56 ,
0xbd6bdab17f670cc2 , 0xa295fb6edc59cc7e ,
0xfea11fbe61e1409f , 0x08f318ebcb10e3c3 ,
0xceb14ffee181302f , 0x06020a08100c0e16 ,
0x49ccdb172e925e67 , 0x51c4f3376ea2663f ,
0x271d6974e84e53cf , 0x3c144450a0786c9c ,
0x58c3e82b56b0730e , 0xa563f2913f57349a ,
0x73da954f9ee63ced , 0xe75d3469d2d38e35 ,
0xe15f3e61c2df8023 , 0x79dc8b57aef22ed7 ,
0x877d94e9cf136e48 , 0x4acdde132694596c ,
0x817f9ee1df1f605e , 0xee5a2f75eac19b04 ,
0xb46cc1ad477519f3 , 0xe45c316ddad5893e ,
0x04f70cfbeb08ffef , 0x6a26be982dd4f247 ,
0x1cff24dbab38c7b7 , 0x2aed7e933b54b911 ,
0x25e86f87134aa236 , 0xba9dd34e9c69f426 ,
0xb16fcea15f7f10ee , 0x8f8e8c0204038d8b ,
0x2b197d64c8564fe3 , 0xfda01aba69e74794 ,
0x0df017e7d31aeade , 0x8689971e3c1198ba ,
0x110f333c78222d69 , 0x09071b1c38121531 ,
0xecaf298611c56afd , 0x10fb30cb8b20db9b ,
0x1808282040303858 , 0x3f154154a87e6b97 ,
0x170d3934682e237f , 0x0c04141020181c2c ,
0x030105040806070b , 0xac64e98d074521ab ,
0x7cdf845bb6f827ca , 0x9a76b3c597295f0d ,
0x8b7980f9ef0b7264 , 0x7add8e53a6f429dc ,
0x473dc9f4f58eb3b2 , 0x3a164e58b074628a ,
0x413fc3fce582bda4 , 0x5937ebdca5b285fc ,
0xb76dc4a94f731ef8 , 0x4838d8e0dd90a895 ,
0xd6b967dea1b10877 , 0x9573a2d1bf37442a ,
0x26e96a831b4ca53d , 0x5f35e1d4b5be8bea ,
0xff551c4992e3b66d , 0x9371a8d9af3b4a3c ,
0x8d7b8af1ff077c72 , 0x898c860a140f839d ,
0x9672a7d5b7314321 , 0x8588921a34179fb1 ,
0x07f609ffe30ef8e4 , 0x7e2a82a84dfcd633 ,
0x423ec6f8ed84baaf , 0xe25e3b65cad98728 ,
0x6927bb9c25d2f54c , 0xca4643050a89cfc0 ,
0x140c3c3060282474 , 0xaf65ec890f4326a0 ,
0xb868d5bd676d05df , 0xa361f8992f5b3a8c ,
0x05030f0c180a091d , 0x5ec1e22346bc7d18 ,
0xf957164182efb87b , 0x67d6a97ffece1899 ,
```

```
0x76d99a4386ec35f0 , 0xe858257dfacd9512 ,
0x75d89f478eea32fb , 0xaa66e38517492fbd ,
0x64d7ac7bf6c81f92 , 0x4e3ad2e8cd9ca683 ,
0x45c8cf070e8a424b , 0x443cccf0fd88b4b9 ,
0x13fa35cf8326dc90 , 0xa796f462c453c563 ,
0xf4a701a651f552a5 , 0xb598c25ab477ef01 ,
0x29ec7b973352be1a , 0xd5b862daa9b70f7c ,
0x54c7fc3b76a86f22 , 0xefae2c8219c36df6 ,
0xbb69d0b96f6b02d4 , 0xdd4b7a3162a7ecbf ,
0xe0ab3d9631dd76d1 , 0xe6a9379e21d178c7 ,
0xa967e6811f4f28b6 , 0x1e0a2228503c364e ,
0xc9474601028fc8cb , 0x0bf21defc316e4c8 ,
0xc2b55beec1992c03 , 0x6622aa880dccee6b ,
0x32e556b37b648149 , 0x2fee719f235eb00c ,
0xdfbe7cc299a31d46 , 0x7d2b87ac45fad138 ,
0x9e81bf3e7c21a0e2 , 0x36125a48906c7ea6 ,
0x9883b5366c2daef4 , 0x2d1b776cd85a41f5 ,
0x120e363870242a62 , 0x6523af8c05cae960 ,
0x02f506f3fb04f1f9 , 0xcf454c091283c6dd ,
0x6321a58415c6e776 , 0x4fced11f3e9e5071 ,
0xdb49703972abe2a9 , 0x742c9cb07de8c409 ,
0x16f93ac39b2cd58d , 0x37e659bf636e8854 ,
0xc7b654e2d993251e , 0x782888a05df0d825 ,
0x39174b5cb8726581 , 0x9b82b032642ba9ff ,
0x2e1a7268d05c46fe , 0x808b9d162c1d96ac ,
0x1ffe21dfa33ec0bc , 0x838a9812241b91a7 ,
0x1b092d2448363f53 , 0x46c9ca03068c4540 ,
0x9487a1264c35b2d8 , 0xd24e6b254ab9f798 ,
0x3ee142a35b7c9d65 , 0x722e96b86de4ca1f ,
0x31e453b773628642 , 0x3de047a7537a9a6e ,
0x20eb608b0b40ab2b , 0xad90ea7af447d759 ,
0xf1a40eaa49ff5bb8 , 0x221e6678f0445ad2 ,
0x9285ab2e5c39bcce , 0xa060fd9d275d3d87 ,
0x0000000000000000 , 0x6f25b19435defb5a ,
0x01f403f7f302f6f2 , 0x0ef112e3db1cedd5 ,
0xa194fe6ad45fcb75 , 0x1d0b272c583a3145 ,
0x34e75cbb6b688f5f , 0x9f75bcc98f235610 ,
0x2cef749b2b58b707 , 0x5c34e4d0bdb88ce1 ,
0x5331f5c495a697c6 , 0x61d4a377eec2168f ,
0x6dd0b767ceda0aa3 , 0x9786a4224433b5d3 ,
0x827e9be5d7196755 , 0xeaad238e01c964eb ,
0x1afd2ed3bb34c9a1 , 0x7b298da455f6df2e ,
0x5030f0c09da090cd , 0x4d3bd7ecc59aa188 ,
0xbc9fd9468c65fa30 , 0x15f83fc7932ad286 ,
0x57c6f93f7eae6829 , 0x35135f4c986a79ad ,
0x0a061e183014123a , 0x0f051114281e1b27 ,
```

```
        0x52c5f63366a46134, 0x33115544886677bb,
        0x9977b6c19f2f5806, 0x847c91edc7156943,
        0x8e7a8ff5f7017b79, 0x887885fde70d756f,
        0x5a36eed8adb482f7, 0x241c6c70e04854c4,
        0x4b39dde4d596af9e, 0xeb592079f2cb9219,
        0x28187860c05048e8, 0xfa5613458ae9bf70,
        0xc8b345f6f18d3e39, 0xcdb04afae9873724,
        0x6c24b4903dd8fc51, 0x6020a0801dc0e07d,
        0xcbb240f2f98b3932, 0xab92e072e44bd94f,
        0xf8a315b671ed4e89, 0x5dc0e7274eba7a13,
        0xcc44490d1a85c1d6, 0xa662f79537513391,
        0x30105040806070b0, 0xc1b45eeac99f2b08,
        0x9184ae2a543fbbc5, 0xc54352112297d4e7,
        0xa893e576ec4dde44, 0x5bc2ed2f5eb67405,
        0xde4a7f356aa1ebb4, 0xdabd73ce81a9145b,
        0x8c8f89060c058a80, 0x772d99b475eec302,
        0xd9bc76ca89af1350, 0xb99cd64a946ff32d,
        0xbe6adfb577610bc9, 0xc0405d1d3a9dddfa,
        0x4ccfd41b3698577a, 0xfba210b279eb4982,
        0x9d80ba3a7427a7e9, 0xd14f6e2142bff093,
        0x211f637cf8425dd9, 0x43cac50f1e864c5d,
        0xe3aa389239db71da, 0xc64257152a91d3ec,
};

const block T2[256] = {
        0xd268bad36a01bbb9, 0x4d1954fc66b1e59a,
        0xbc932f7114cde265, 0xcdb9749c1b512587,
        0x510253f557a4f7a2, 0x6bb8d368be03d0d6,
        0x6fbdd26bb504d6de, 0x29644dd785feb352,
        0x5d0d50f04aadfdba, 0x8a26ace9e063cf09,
        0x0e838d8a9684091c, 0xc679bfdc4d1aa591,
        0xddad7090374d3da7, 0x550752f65ca3f1aa,
        0x52c89ab317e17ba4, 0x2d614cd48ef9b55a,
        0x8f65ea2320ac4603, 0x73a6d5628411c4e6,
        0x66f197a468c255cc, 0x63b2d16ea80ddcc6,
        0xccff3355d099aa85, 0x590851f341aafbb2,
        0x712a5bed0f9cc7e2, 0xa204a6f7ae55f359,
        0x5f81de7fc120febe, 0x3d7548d8a2e5ad7a,
        0x9a32a8e5cc7fd729, 0x5ec799b60ae871bc,
        0x4b90db70e63be096, 0xc8fa3256db9eac8d,
        0xe651b7c4152295d1, 0xd72bfc19aace32b3,
        0xab48e3387393704b, 0x42dc9ebf3bfd6384,
        0x7eef91ae52d041fc, 0x56cd9bb01ce67dac,
        0xaf4de23b78947643, 0xd66dbbd06106bdb1,
        0x195841c3f1da9b32, 0xa5cb6eb2e5177957,
        0xae0ba5f2b35cf941, 0x0bc0cb40564b8016,
```

```
0xb1da6bbdc20c677f, 0x6efb95a27ecc59dc,
0xbe1fa1fe9f40e161, 0xeb18f308c3e310cb,
0xfe4fb1ce2f3081e1, 0x080a0206160e0c10,
0x17dbcc49675e922e, 0x37f3c4513f66a26e,
0x74691d27cf534ee8, 0x5044143c9c6c78a0,
0x2be8c3580e73b056, 0x91f263a59a34573f,
0x4f95da73ed3ce69e, 0x69345de7358ed3d2,
0x613e5fe12380dfc2, 0x578bdc79d72ef2ae,
0xe9947d87486e13cf, 0x13decd4a6c599426,
0xe19e7f815e601fdf, 0x752f5aee049bc1ea,
0xadc16cb4f3197547, 0x6d315ce43e89d5da,
0xfb0cf704efff08eb, 0x98be266a47f2d42d,
0xdb24ff1cb7c738ab, 0x937eed2a11b9543b,
0x876fe82536a24a13, 0x4ed39dba26f4699c,
0xa1ce6fb1ee107f5f, 0x028c8e8f8b8d0304,
0x647d192be34f56c8, 0xba1aa0fd9447e769,
0xe717f00ddeea1ad3, 0x1e978986ba98113c,
0x3c330f11692d2278, 0x1c1b070931151238,
0x8629afecfd6ac511, 0xcb30fb109bdb208b,
0x2028081858383040, 0x5441153f976b7ea8,
0x34390d177f232e68, 0x1014040c2c1c1820,
0x040501030b070608, 0x8de964acab214507,
0x5b84df7cca27f8b6, 0xc5b3769a0d5f2997,
0xf980798b64720bef, 0x538edd7adc29f4a6,
0xf4c93d47b2b38ef5, 0x584e163a8a6274b0,
0xfcc33f41a4bd82e5, 0xdceb3759fc85b2a5,
0xa9c46db7f81e734f, 0xe0d8384895a890dd,
0xde67b9d67708b1a1, 0xd1a273952a4437bf,
0x836ae9263da54c1b, 0xd4e1355fea8bbeb5,
0x491c55ff6db6e392, 0xd9a871933c4a3baf,
0xf18a7b8d727c07ff, 0x0a868c899d830f14,
0xd5a77296214331b7, 0x1a928885b19f1734,
0xff09f607e4f80ee3, 0xa8822a7e33d6fc4d,
0xf8c63e42afba84ed, 0x653b5ee22887d9ca,
0x9cbb27694cf5d225, 0x054346cac0cf890a,
0x303c0c1474242860, 0x89ec65afa026430f,
0xbdd568b8df056d67, 0x99f861a38c3a5b2f,
0x0c0f03051d090a18, 0x23e2c15e187dbc46,
0x411657f97bb8ef82, 0x7fa9d6679918cefe,
0x439ad976f035ec86, 0x7d2558e81295cdfa,
0x479fd875fb32ea8e, 0x85e366aabd2f4917,
0x7bacd764921fc8f6, 0xe8d23a4e83a69ccd,
0x07cfc8454b428a0e, 0xf0cc3c44b9b488fd,
0xcf35fa1390dc2683, 0x62f496a763c553c4,
0xa601a7f4a552f551, 0x5ac298b501ef77b4,
0x977bec291abe5233, 0xda62b8d57c0fb7a9,
```

```
0x3bfcc754226fa876 , 0x822caeeff66dc319 ,
0xb9d069bbd4026b6f , 0x317a4bddbfeca762 ,
0x963dabe0d176dd31 , 0x9e37a9e6c778d121 ,
0x81e667a9b6284f1f , 0x28220a1e4e363c50 ,
0x014647c9cbc88f02 , 0xef1df20bc8e416c3 ,
0xee5bb5c2032c99c1 , 0x88aa22666beecc0d ,
0xb356e5324981647b , 0x9f71ee2f0cb05e23 ,
0xc27cbedf461da399 , 0xac872b7d38d1fa45 ,
0x3ebf819ee2a0217c , 0x485a1236a67e6c90 ,
0x36b58398f4ae2d6c , 0x6c771b2df5415ad8 ,
0x38360e12622a2470 , 0x8caf236560e9ca05 ,
0xf306f502f9f104fb , 0x094c45cfddc68312 ,
0x84a5216376e7c615 , 0x1fd1ce4f71509e3e ,
0x397049dba9e2ab72 , 0xb09c2c7409c4e87d ,
0xc33af9168dd52c9b , 0xbf59e63754886e63 ,
0xe254b6c71e2593d9 , 0xa088287825d8f05d ,
0x5c4b1739816572b8 , 0x32b0829bffa92b64 ,
0x68721a2efe465cd0 , 0x169d8b80ac961d2c ,
0xdf21fe1fbcc03ea3 , 0x12988a83a7911b24 ,
0x242d091b533f3648 , 0x03cac94640458c06 ,
0x26a18794d8b2354c , 0x256b4ed298f7b94a ,
0xa342e13e659d7c5b , 0xb8962e721fcae46d ,
0xb753e43142866273 , 0xa747e03d6e9a7a53 ,
0x8b60eb202bab400b , 0x7aea90ad59d747f4 ,
0xaa0ea4f1b85bff49 , 0x78661e22d25a44f0 ,
0x2eab8592cebc395c , 0x9dfd60a0873d5d27 ,
0x0000000000000000 , 0x94b1256f5afbde35 ,
0xf703f401f2f602f3 , 0xe312f10ed5ed1cdb ,
0x6afe94a175cb5fd4 , 0x2c270b1d45313a58 ,
0xbb5ce7345f8f686b , 0xc9bc759f1056238f ,
0x9b74ef2c07b7582b , 0xd0e4345ce18cb8bd ,
0xc4f53153c697a695 , 0x77a3d4618f16c2ee ,
0x67b7d06da30adace , 0x22a48697d3b53344 ,
0xe59b7e82556719d7 , 0x8e23adeaeb64c901 ,
0xd32efd1aa1c934bb , 0xa48d297b2edff655 ,
0xc0f03050cd90a09d , 0xecd73b4d88a19ac5 ,
0x46d99fbc30fa658c , 0xc73ff81586d22a93 ,
0x3ff9c6572968ae7e , 0x4c5f1335ad796a98 ,
0x181e060a3a121430 , 0x1411050f271b1e28 ,
0x33f6c5523461a466 , 0x44551133bb776688 ,
0xc1b6779906582f9f , 0xed917c84436915c7 ,
0xf58f7a8e797b01f7 , 0xfd8578886f750de7 ,
0xd8ee365af782b4ad , 0x706c1c24c45448e0 ,
0xe4dd394b9eaf96d5 , 0x792059eb1992cbf2 ,
0x60781828e84850c0 , 0x451356fa70bfe98a ,
0xf645b3c8393e8df1 , 0xfa4ab0cd243787e9 ,
```

```
        0x90b4246c51fcd83d, 0x80a020607de0c01d,
        0xf240b2cb32398bf9, 0x72e092ab4fd94be4,
        0xb615a3f8894eed71, 0x27e7c05d137aba4e,
        0x0d4944ccd6c1851a, 0x95f762a691335137,
        0x40501030b0706080, 0xea5eb4c1082b9fc9,
        0x2aae8491c5bb3f54, 0x115243c5e7d49722,
        0x76e593a844de4dec, 0x2fedc25b0574b65e,
        0x357f4adeb4eba16a, 0xce73bdda5b14a981,
        0x06898f8c808a050c, 0xb4992d7702c3ee75,
        0xca76bcd95013af89, 0x4ad69cb92df36f94,
        0xb5df6abec90b6177, 0x1d5d40c0fadd9d3a,
        0x1bd4cf4c7a579836, 0xb210a2fb8249eb79,
        0x3aba809de9a72774, 0x216e4fd193f0bf42,
        0x7c631f21d95d42f8, 0x0fc5ca435d4c861e,
        0x9238aae3da71db39, 0x155742c6ecd3912a,
};

const block T3[256] = {
        0x68d2d3ba016ab9bb, 0x194dfc54b1669ae5,
        0x93bc712fcd1465e2, 0xb9cd9c74511b8725,
        0x0251f553a457a2f7, 0xb86b68d303bed6d0,
        0xbd6f6bd204b5ded6, 0x6429d74dfe8552b3,
        0x0d5df050ad4abafd, 0x268ae9ac63e009cf,
        0x830e8a8d84961c09, 0x79c6dcbf1a4d91a5,
        0xaddd90704d37a73d, 0x0755f652a35caaf1,
        0xc852b39ae117a47b, 0x612dd44cf98e5ab5,
        0x658f23eaac200346, 0xa67362d51184e6c4,
        0xf166a497c268cc55, 0xb2636ed10da8c6dc,
        0xffcc553399d085aa, 0x0859f351aa41b2fb,
        0x2a71ed5b9c0fe2c7, 0x04a2f7a655ae59f3,
        0x815f7fde20c1befe, 0x753dd848e5a27aad,
        0x329ae5a87fcc29d7, 0xc75eb699e80abc71,
        0x904b70db3be696e0, 0xfac856329edb8dac,
        0x51e6c4b72215d195, 0x2bd719fcceaab332,
        0x48ab38e393734b70, 0xdc42bf9efd3b8463,
        0xef7eae91d052fc41, 0xcd56b09be61cac7d,
        0x4daf3be294784376, 0x6dd6d0bb0661b1bd,
        0x5819c341daf1329b, 0xcba5b26e17e55779,
        0x0baef2a55cb341f9, 0xc00b40cb4b561680,
        0xdab1bd6b0cc27f67, 0xfb6ea295cc7edc59,
        0x1fbefea1409f61e1, 0x18eb08f3e3c3cb10,
        0x4ffeceb1302fe181, 0x0a0806020e16100c,
        0xdb1749cc5e672e92, 0xf33751c4663f6ea2,
        0x6974271d53cfe84e, 0x44503c146c9ca078,
        0xe82b58c3730e56b0, 0xf291a563349a3f57,
```

```
0x954f73da3ced9ee6 , 0x3469e75d8e35d2d3 ,
0x3e61e15f8023c2df , 0x8b5779dc2ed7aef2 ,
0x94e9877d6e48cf13 , 0xde134acd596c2694 ,
0x9ee1817f605edf1f , 0x2f75ee5a9b04eac1 ,
0xc1adb46c19f34775 , 0x316de45c893edad5 ,
0x0cfb04f7ffefeb08 , 0xbe986a26f2472dd4 ,
0x24db1cffc7b7ab38 , 0x7e932aedb9113b54 ,
0x6f8725e8a236134a , 0xd34eba9df4269c69 ,
0xcea1b16f10ee5f7f , 0x8c028f8e8d8b0403 ,
0x7d642b194fe3c856 , 0x1abafda0479469e7 ,
0x17e70df0eaded31a , 0x971e868998ba3c11 ,
0x333c110f2d697822 , 0x1b1c090715313812 ,
0x2986ecaf6afd11c5 , 0x30cb10fbdb9b8b20 ,
0x2820180838584030 , 0x41543f156b97a87e ,
0x3934170d237f682e , 0x14100c041c2c2018 ,
0x05040301070b0806 , 0xe98dac6421ab0745 ,
0x845b7cdf27cab6f8 , 0xb3c59a765f0d9729 ,
0x80f98b797264ef0b , 0x8e537add29dca6f4 ,
0xc9f4473db3b2f58e , 0x4e583a16628ab074 ,
0xc3fc413fbda4e582 , 0xebdc593785fca5b2 ,
0xc4a9b76d1ef84f73 , 0xd8e04838a895dd90 ,
0x67ded6b90877a1b1 , 0xa2d19573442abf37 ,
0x6a8326e9a53d1b4c , 0xe1d45f358beab5be ,
0x1c49ff55b66d92e3 , 0xa8d993714a3caf3b ,
0x8af18d7b7c72ff07 , 0x860a898c839d140f ,
0xa7d596724321b731 , 0x921a85889fb13417 ,
0x09ff07f6f8e4e30e , 0x82a87e2ad6334dfc ,
0xc6f8423ebaafed84 , 0x3b65e25e8728cad9 ,
0xbb9c6927f54c25d2 , 0x4305ca46cfc00a89 ,
0x3c30140c24746028 , 0xec89af6526a00f43 ,
0xd5bdb86805df676d , 0xf899a3613a8c2f5b ,
0x0f0c0503091d180a , 0xe2235ec17d1846bc ,
0x1641f957b87b82ef , 0xa97f67d61899fece ,
0x9a4376d935f086ec , 0x257de8589512facd ,
0x9f4775d832fb8eea , 0xe385aa662fbd1749 ,
0xac7b64d71f92f6c8 , 0xd2e84e3aa683cd9c ,
0xcf0745c8424b0e8a , 0xccf0443cb4b9fd88 ,
0x35cf13fadc908326 , 0xf462a796c563c453 ,
0x01a6f4a752a551f5 , 0xc25ab598ef01b477 ,
0x7b9729ecbe1a3352 , 0x62dad5b80f7ca9b7 ,
0xfc3b54c76f2276a8 , 0x2c82efae6df619c3 ,
0xd0b9bb6902d46f6b , 0x7a31dd4becbf62a7 ,
0x3d96e0ab76d131dd , 0x379ee6a978c721d1 ,
0xe681a96728b61f4f , 0x22281e0a364e503c ,
0x4601c947c8cb028f , 0x1def0bf2e4c8c316 ,
0x5beec2b52c03c199 , 0xaa886622ee6b0dcc ,
```

```
0x56b332e581497b64 , 0x719f2feeb00c235e ,
0x7cc2dfbe1d4699a3 , 0x87ac7d2bd13845fa ,
0xbf3e9e81a0e27c21 , 0x5a4836127ea6906c ,
0xb5369883aef46c2d , 0x776c2d1b41f5d85a ,
0x3638120e2a627024 , 0xaf8c6523e96005ca ,
0x06f302f5f1f9fb04 , 0x4c09cf45c6dd1283 ,
0xa5846321e77615c6 , 0xd11f4fce50713e9e ,
0x7039db49e2a972ab , 0x9cb0742cc4097de8 ,
0x3ac316f9d58d9b2c , 0x59bf37e68854636e ,
0x54e2c7b6251ed993 , 0x88a07828d8255df0 ,
0x4b5c39176581b872 , 0xb0329b82a9ff642b ,
0x72682e1a46fed05c , 0x9d16808b96ac2c1d ,
0x21df1ffec0bca33e , 0x9812838a91a7241b ,
0x2d241b093f534836 , 0xca0346c94540068c ,
0xa1269487b2d84c35 , 0x6b25d24ef7984ab9 ,
0x42a33ee19d655b7c , 0x96b8722eca1f6de4 ,
0x53b731e486427362 , 0x47a73de09a6e537a ,
0x608b20ebab2b0b40 , 0xea7aad90d759f447 ,
0x0eaaf1a45bb849ff , 0x6678221e5ad2f044 ,
0xab2e9285bcce5c39 , 0xfd9da0603d87275d ,
0x0000000000000000 , 0xb1946f25fb5a35de ,
0x03f701f4f6f2f302 , 0x12e30ef1edd5db1c ,
0xfe6aa194cb75d45f , 0x272c1d0b3145583a ,
0x5cbb34e78f5f6b68 , 0xbcc99f7556108f23 ,
0x749b2cefb7072b58 , 0xe4d05c348ce1bdb8 ,
0xf5c4533197c695a6 , 0xa37761d4168feec2 ,
0xb7676dd00aa3ceda , 0xa4229786b5d34433 ,
0x9be5827e6755d719 , 0x238eeaad64eb01c9 ,
0x2ed31afdc9a1bb34 , 0x8da47b29df2e55f6 ,
0xf0c0503090cd9da0 , 0xd7ec4d3ba188c59a ,
0xd946bc9ffa308c65 , 0x3fc715f8d286932a ,
0xf93f57c668297eae , 0x5f4c351379ad986a ,
0x1e180a06123a3014 , 0x11140f051b27281e ,
0xf63352c5613466a4 , 0x5544331177bb8866 ,
0xb6c1997758069f2f , 0x91ed847c6943c715 ,
0x8ff58e7a7b79f701 , 0x85fd8878756fe70d ,
0xeed85a3682f7adb4 , 0x6c70241c54c4e048 ,
0xdde44b39af9ed596 , 0x2079eb599219f2cb ,
0x7860281848e8c050 , 0x1345fa56bf708ae9 ,
0x45f6c8b33e39f18d , 0x4afacdb03724e987 ,
0xb4906c24fc513dd8 , 0xa0806020e07d1dc0 ,
0x40f2cbb23932f98b , 0xe072ab92d94fe44b ,
0x15b6f8a34e8971ed , 0xe7275dc07a134eba ,
0x490dcc44c1d61a85 , 0xf795a66233913751 ,
0x5040301070b08060 , 0x5eeac1b42b08c99f ,
0xae2a9184bbc5543f , 0x5211c543d4e72297 ,
```

```
        0xe576a893de44ec4d, 0xed2f5bc274055eb6,
        0x7f35de4aebb46aa1, 0x73cedabd145b81a9,
        0x89068c8f8a800c05, 0x99b4772dc30275ee,
        0x76cad9bc135089af, 0xd64ab99cf32d946f,
        0xdfb5be6a0bc97761, 0x5d1dc040ddfa3a9d,
        0xd41b4ccf577a3698, 0x10b2fba2498279eb,
        0xba3a9d80a7e97427, 0x6e21d14ff09342bf,
        0x637c211f5dd9f842, 0xc50f43ca4c5d1e86,
        0x3892e3aa71da39db, 0x5715c642d3ec2a91,
};


const block T4[256] = {
        0xbbb96a01bad3d268, 0xe59a66b154fc4d19,
        0xe26514cd2f71bc93, 0x25871b51749ccdb9,
        0xf7a257a453f55102, 0xd0d6be03d3686bb8,
        0xd6deb504d26b6fbd, 0xb35285fe4dd72964,
        0xfdba4aad50f05d0d, 0xcf09e063ace98a26,
        0x091c96848d8a0e83, 0xa5914d1abfdcc679,
        0x3da7374d7090ddad, 0xf1aa5ca352f65507,
        0x7ba417e19ab352c8, 0xb55a8ef94cd42d61,
        0x460320acea238f65, 0xc4e68411d56273a6,
        0x55cc68c297a466f1, 0xdcc6a80dd16e63b2,
        0xaa85d0993355ccff, 0xfbb241aa51f35908,
        0xc7e20f9c5bed712a, 0xf359ae55a6f7a204,
        0xfebec120de7f5f81, 0xad7aa2e548d83d75,
        0xd729cc7fa8e59a32, 0x71bc0ae899b65ec7,
        0xe096e63bdb704b90, 0xac8ddb9e3256c8fa,
        0x95d11522b7c4e651, 0x32b3aacefc19d72b,
        0x704b7393e338ab48, 0x63843bfd9ebf42dc,
        0x41fc52d091ae7eef, 0x7dac1ce69bb056cd,
        0x76437894e23baf4d, 0xbdb16106bbd0d66d,
        0x9b32f1da41c31958, 0x7957e5176eb2a5cb,
        0xf941b35ca5f2ae0b, 0x8016564bcb400bc0,
        0x677fc20c6bbdb1da, 0x59dc7ecc95a26efb,
        0xe1619f40a1febe1f, 0x10cbc3e3f308eb18,
        0x81e12f30b1cefe4f, 0x0c10160e0206080a,
        0x922e675ecc4917db, 0xa26e3f66c45137f3,
        0x4ee8cf531d277469, 0x78a09c6c143c5044,
        0xb0560e73c3582be8, 0x573f9a3463a591f2,
        0xe69eed3cda734f95, 0xd3d2358e5de76934,
        0xdfc223805fe1613e, 0xf2aed72edc79578b,
        0x13cf486e7d87e994, 0x94266c59cd4a13de,
        0x1fdf5e607f81e19e, 0xc1ea049b5aee752f,
        0x7547f3196cb4adc1, 0xd5da3e895ce46d31,
        0x08ebefffff704fb0c, 0xd42d47f2266a98be,
```

```
0x38abb7c7ff1cdb24 , 0x543b11b9ed2a937e ,
0x4a1336a2e825876f , 0x699c26f49dba4ed3 ,
0x7f5fee106fb1a1ce , 0x03048b8d8e8f028c ,
0x56c8e34f192b647d , 0xe7699447a0fdba1a ,
0x1ad3deeaf00de717 , 0x113cba9889861e97 ,
0x2278692d0f113c33 , 0x1238311507091c1b ,
0xc511fd6aafec8629 , 0x208b9bdbfb10cb30 ,
0x3040583808182028 , 0x7ea8976b153f5441 ,
0x2e687f230d173439 , 0x18202c1c040c1014 ,
0x06080b0701030405 , 0x4507ab2164ac8de9 ,
0xf8b6ca27df7c5b84 , 0x29970d5f769ac5b3 ,
0x0bef6472798bf980 , 0xf4a6dc29dd7a538e ,
0x8ef5b2b33d47f4c9 , 0x74b08a62163a584e ,
0x82e5a4bd3f41fcc3 , 0xb2a5fc853759dceb ,
0x734ff81e6db7a9c4 , 0x90dd95a83848e0d8 ,
0xb1a17708b9d6de67 , 0x37bf2a447395d1a2 ,
0x4c1b3da5e926836a , 0xbeb5ea8b355fd4e1 ,
0xe3926db655ff491c , 0x3baf3c4a7193d9a8 ,
0x07ff727c7b8df18a , 0x0f149d838c890a86 ,
0x31b721437296d5a7 , 0x1734b19f88851a92 ,
0x0ee3e4f8f607ff09 , 0xfc4d33d62a7ea882 ,
0x84edafba3e42f8c6 , 0xd9ca28875ee2653b ,
0xd2254cf527699cbb , 0x890ac0cf46ca0543 ,
0x286074240c14303c , 0x430fa02665af89ec ,
0x6d67df0568b8bdd5 , 0x5b2f8c3a61a399f8 ,
0x0a181d0903050c0f , 0xbc46187dc15e23e2 ,
0xef827bb857f94116 , 0xcefe9918d6677fa9 ,
0xec86f035d976439a , 0xcdfa129558e87d25 ,
0xea8efb32d875479f , 0x4917bd2f66aa85e3 ,
0xc8f6921fd7647bac , 0x9ccd83a63a4ee8d2 ,
0x8a0e4b42c84507cf , 0x88fdb9b43c44f0cc ,
0x268390dcfa13cf35 , 0x53c463c596a762f4 ,
0xf551a552a7f4a601 , 0x77b401ef98b55ac2 ,
0x52331abeec29977b , 0xb7a97c0fb8d5da62 ,
0xa876226fc7543bfc , 0xc319f66daeef822c ,
0x6b6fd40269bbb9d0 , 0xa762bfec4bdd317a ,
0xdd31d176abe0963d , 0xd121c778a9e69e37 ,
0x4f1fb62867a981e6 , 0x3c504e360a1e2822 ,
0x8f02cbc847c90146 , 0x16c3c8e4f20bef1d ,
0x99c1032cb5c2ee5b , 0xcc0d6bee226688aa ,
0x647b4981e532b356 , 0x5e230cb0ee2f9f71 ,
0xa399461dbedfc27c , 0xfa4538d12b7dac87 ,
0x217ce2a0819e3ebf , 0x6c90a67e1236485a ,
0x2d6cf4ae839836b5 , 0x5ad8f5411b2d6c77 ,
0x2470622a0e123836 , 0xca0560e923658caf ,
0x04fbf9f1f502f306 , 0x8312ddc645cf094c ,
```

```
0xc61576e7216384a5 , 0x9e3e7150ce4f1fd1 ,
0xab72a9e249db3970 , 0xe87d09c42c74b09c ,
0x2c9b8dd5f916c33a , 0x6e635488e637bf59 ,
0x93d91e25b6c7e254 , 0xf05d25d82878a088 ,
0x72b8816517395c4b , 0x2b64ffa9829b32b0 ,
0x5cd0fe461a2e6872 , 0x1d2cac968b80169d ,
0x3ea3bcc0fe1fdf21 , 0x1b24a7918a831298 ,
0x3648533f091b242d , 0x8c064045c94603ca ,
0x354cd8b2879426a1 , 0xb94a98f74ed2256b ,
0x7c5b659de13ea342 , 0xe46d1fca2e72b896 ,
0x62734286e431b753 , 0x7a536e9ae03da747 ,
0x400b2babeb208b60 , 0x47f459d790ad7aea ,
0xff49b85ba4f1aa0e , 0x44f0d25a1e227866 ,
0x395ccebc85922eab , 0x5d27873d60a09dfd ,
0x0000000000000000 , 0xde355afb256f94b1 ,
0x02f3f2f6f401f703 , 0x1cdbd5edf10ee312 ,
0x5fd475cb94a16afe , 0x3a5845310b1d2c27 ,
0x686b5f8fe734bb5c , 0x238f1056759fc9bc ,
0x582b07b7ef2c9b74 , 0xb8bde18c345cd0e4 ,
0xa695c6973153c4f5 , 0xc2ee8f16d46177a3 ,
0xdacea30ad06d67b7 , 0x3344d3b5869722a4 ,
0x19d755677e82e59b , 0xc901eb64adea8e23 ,
0x34bba1c9fd1ad32e , 0xf6552edf297ba48d ,
0xa09dcd903050c0f0 , 0x9ac588a13b4decd7 ,
0x658c30fa9fbc46d9 , 0x2a9386d2f815c73f ,
0xae7e2968c6573ff9 , 0x6a98ad7913354c5f ,
0x14303a12060a181e , 0x1e28271b050f1411 ,
0xa4663461c55233f6 , 0x6688bb7711334455 ,
0x2f9f06587799c1b6 , 0x15c743697c84ed91 ,
0x01f7797b7a8ef58f , 0x0de76f757888fd85 ,
0xb4adf782365ad8ee , 0x48e0c4541c24706c ,
0x96d59eaf394be4dd , 0xcbf2199259eb7920 ,
0x50c0e84818286078 , 0xe98a70bf56fa4513 ,
0x8df1393eb3c8f645 , 0x87e92437b0cdfa4a ,
0xd83d51fc246c90b4 , 0xc01d7de0206080a0 ,
0x8bf93239b2cbf240 , 0x4be44fd992ab72e0 ,
0xed71894ea3f8b615 , 0xba4e137ac05d27e7 ,
0x851ad6c144cc0d49 , 0x5137913362a695f7 ,
0x6080b07010304050 , 0x9fc9082bb4c1ea5e ,
0x3f54c5bb84912aae , 0x9722e7d443c51152 ,
0x4dec44de93a876e5 , 0xb65e0574c25b2fed ,
0xa16ab4eb4ade357f , 0xa9815b14bddace73 ,
0x050c808a8f8c0689 , 0xee7502c32d77b499 ,
0xaf895013bcd9ca76 , 0x6f942df39cb94ad6 ,
0x6177c90b6abeb5df , 0x9d3afadd40c01d5d ,
0x98367a57cf4c1bd4 , 0xeb798249a2fbb210 ,
```

```
        0x2774e9a7809d3aba, 0xbf4293f04fd1216e,
        0x42f8d95d1f217c63, 0x861e5d4cca430fc5,
        0xdb39da71aae39238, 0x912aecd342c61557,
};

const block T5[256] = {
        0xb9bb016ad3ba68d2, 0x9ae5b166fc54194d,
        0x65e2cd14712f93bc, 0x8725511b9c74b9cd,
        0xa2f7a457f5530251, 0xd6d003be68d3b86b,
        0xded604b56bd2bd6f, 0x52b3fe85d74d6429,
        0xbafdad4af0500d5d, 0x09cf63e0e9ac268a,
        0x1c0984968a8d830e, 0x91a51a4ddcbf79c6,
        0xa73d4d379070addd, 0xaaf1a35cf6520755,
        0xa47be117b39ac852, 0x5ab5f98ed44c612d,
        0x0346ac2023ea658f, 0xe6c4118462d5a673,
        0xcc55c268a497f166, 0xc6dc0da86ed1b263,
        0x85aa99d05533ffcc, 0xb2fbaa41f3510859,
        0xe2c79c0fed5b2a71, 0x59f355aef7a604a2,
        0xbefe20c17fde815f, 0x7aade5a2d848753d,
        0x29d77fcce5a8329a, 0xbc71e80ab699c75e,
        0x96e03be670db904b, 0x8dac9edb5632fac8,
        0xd1952215c4b751e6, 0xb332ceaa19fc2bd7,
        0x4b70937338e348ab, 0x8463fd3bbf9edc42,
        0xfc41d052ae91ef7e, 0xac7de61cb09bcd56,
        0x437694783be24daf, 0xb1bd0661d0bb6dd6,
        0x329bdaf1c3415819, 0x577917e5b26ecba5,
        0x41f95cb3f2a50bae, 0x16804b5640cbc00b,
        0x7f670cc2bd6bdab1, 0xdc59cc7ea295fb6e,
        0x61e1409ffea11fbe, 0xcb10e3c308f318eb,
        0xe181302fceb14ffe, 0x100c0e1606020a08,
        0x2e925e6749ccdb17, 0x6ea2663f51c4f337,
        0xe84e53cf271d6974, 0xa0786c9c3c144450,
        0x56b0730e58c3e82b, 0x3f57349aa563f291,
        0x9ee63ced73da954f, 0xd2d38e35e75d3469,
        0xc2df8023e15f3e61, 0xaef22ed779dc8b57,
        0xcf136e48877d94e9, 0x2694596c4acdde13,
        0xdf1f605e817f9ee1, 0xeac19b04ee5a2f75,
        0x477519f3b46cc1ad, 0xdad5893ee45c316d,
        0xeb08ffef04f70cfb, 0x2dd4f2476a26be98,
        0xab38c7b71cff24db, 0x3b54b9112aed7e93,
        0x134aa23625e86f87, 0x9c69f426ba9dd34e,
        0x5f7f10eeb16fcea1, 0x04038d8b8f8e8c02,
        0xc8564fe32b197d64, 0x69e74794fda01aba,
        0xd31aeade0df017e7, 0x3c1198ba8689971e,
        0x78222d69110f333c, 0x3812153109071b1c,
        0x11c56afdecaf2986, 0x8b20db9b10fb30cb,
```

```
0x4030385818082820 , 0xa87e6b973f154154 ,
0x682e237f170d3934 , 0x20181c2c0c041410 ,
0x0806070b03010504 , 0x074521abac64e98d ,
0xb6f827ca7cdf845b , 0x97295f0d9a76b3c5 ,
0xef0b72648b7980f9 , 0xa6f429dc7add8e53 ,
0xf58eb3b2473dc9f4 , 0xb074628a3a164e58 ,
0xe582bda4413fc3fc , 0xa5b285fc5937ebdc ,
0x4f731ef8b76dc4a9 , 0xdd90a8954838d8e0 ,
0xa1b10877d6b967de , 0xbf37442a9573a2d1 ,
0x1b4ca53d26e96a83 , 0xb5be8bea5f35e1d4 ,
0x92e3b66dff551c49 , 0xaf3b4a3c9371a8d9 ,
0xff077c728d7b8af1 , 0x140f839d898c860a ,
0xb73143219672a7d5 , 0x34179fb18588921a ,
0xe30ef8e407f609ff , 0x4dfcd6337e2a82a8 ,
0xed84baaf423ec6f8 , 0xcad98728e25e3b65 ,
0x25d2f54c6927bb9c , 0x0a89cfc0ca464305 ,
0x60282474140c3c30 , 0x0f4326a0af65ec89 ,
0x676d05dfb868d5bd , 0x2f5b3a8ca361f899 ,
0x180a091d05030f0c , 0x46bc7d185ec1e223 ,
0x82efb87bf9571641 , 0xfece189967d6a97f ,
0x86ec35f076d99a43 , 0xfacd9512e858257d ,
0x8eea32fb75d89f47 , 0x17492fbdaa66e385 ,
0xf6c81f9264d7ac7b , 0xcd9ca6834e3ad2e8 ,
0x0e8a424b45c8cf07 , 0xfd88b4b9443cccf0 ,
0x8326dc9013fa35cf , 0xc453c563a796f462 ,
0x51f552a5f4a701a6 , 0xb477ef01b598c25a ,
0x3352be1a29ec7b97 , 0xa9b70f7cd5b862da ,
0x76a86f2254c7fc3b , 0x19c36df6efae2c82 ,
0x6f6b02d4bb69d0b9 , 0x62a7ecbfdd4b7a31 ,
0x31dd76d1e0ab3d96 , 0x21d178c7e6a9379e ,
0x1f4f28b6a967e681 , 0x503c364e1e0a2228 ,
0x028fc8cbc9474601 , 0xc316e4c80bf21def ,
0xc1992c03c2b55bee , 0x0dccee6b6622aa88 ,
0x7b64814932e556b3 , 0x235eb00c2fee719f ,
0x99a31d46dfbe7cc2 , 0x45fad1387d2b87ac ,
0x7c21a0e29e81bf3e , 0x906c7ea636125a48 ,
0x6c2daef49883b536 , 0xd85a41f52d1b776c ,
0x70242a62120e3638 , 0x05cae9606523af8c ,
0xfb04f1f902f506f3 , 0x1283c6ddcf454c09 ,
0x15c6e7766321a584 , 0x3e9e50714fced11f ,
0x72abe2a9db497039 , 0x7de8c409742c9cb0 ,
0x9b2cd58d16f93ac3 , 0x636e885437e659bf ,
0xd993251ec7b654e2 , 0x5df0d825782888a0 ,
0xb872658139174b5c , 0x642ba9ff9b82b032 ,
0xd05c46fe2e1a7268 , 0x2c1d96ac808b9d16 ,
0xa33ec0bc1ffe21df , 0x241b91a7838a9812 ,
```

```
        0x48363f531b092d24, 0x068c454046c9ca03,
        0x4c35b2d89487a126, 0x4ab9f798d24e6b25,
        0x5b7c9d653ee142a3, 0x6de4ca1f722e96b8,
        0x7362864231e453b7, 0x537a9a6e3de047a7,
        0x0b40ab2b20eb608b, 0xf447d759ad90ea7a,
        0x49ff5bb8f1a40eaa, 0xf0445ad2221e6678,
        0x5c39bcce9285ab2e, 0x275d3d87a060fd9d,
        0x0000000000000000, 0x35defb5a6f25b194,
        0xf302f6f201f403f7, 0xdb1cedd50ef112e3,
        0xd45fcb75a194fe6a, 0x583a31451d0b272c,
        0x6b688f5f34e75cbb, 0x8f2356109f75bcc9,
        0x2b58b7072cef749b, 0xbdb88ce15c34e4d0,
        0x95a697c65331f5c4, 0xeec2168f61d4a377,
        0xceda0aa36dd0b767, 0x4433b5d39786a422,
        0xd7196755827e9be5, 0x01c964ebeaad238e,
        0xbb34c9a11afd2ed3, 0x55f6df2e7b298da4,
        0x9da090cd5030f0c0, 0xc59aa1884d3bd7ec,
        0x8c65fa30bc9fd946, 0x932ad28615f83fc7,
        0x7eae682957c6f93f, 0x986a79ad35135f4c,
        0x3014123a0a061e18, 0x281e1b270f051114,
        0x66a4613452c5f633, 0x886677bb33115544,
        0x9f2f58069977b6c1, 0xc7156943847c91ed,
        0xf7017b798e7a8ff5, 0xe70d756f887885fd,
        0xadb482f75a36eed8, 0xe04854c4241c6c70,
        0xd596af9e4b39dde4, 0xf2cb9219eb592079,
        0xc05048e828187860, 0x8ae9bf70fa561345,
        0xf18d3e39c8b345f6, 0xe9873724cdb04afa,
        0x3dd8fc516c24b490, 0x1dc0e07d6020a080,
        0xf98b3932cbb240f2, 0xe44bd94fab92e072,
        0x71ed4e89f8a315b6, 0x4eba7a135dc0e727,
        0x1a85c1d6cc44490d, 0x37513391a662f795,
        0x806070b030105040, 0xc99f2b08c1b45eea,
        0x543fbbc59184ae2a, 0x2297d4e7c5435211,
        0xec4dde44a893e576, 0x5eb674055bc2ed2f,
        0x6aa1ebb4de4a7f35, 0x81a9145bdabd73ce,
        0x0c058a808c8f8906, 0x75eec302772d99b4,
        0x89af1350d9bc76ca, 0x946ff32db99cd64a,
        0x77610bc9be6adfb5, 0x3a9dddfac0405d1d,
        0x3698577a4ccfd41b, 0x79eb4982fba210b2,
        0x7427a7e99d80ba3a, 0x42bff093d14f6e21,
        0xf8425dd9211f637c, 0x1e864c5d43cac50f,
        0x39db71dae3aa3892, 0x2a91d3ecc6425715,
};

const block T6[256] = {
        0x6a01bbb9d268bad3, 0x66b1e59a4d1954fc,
```

```
0x14cde265bc932f71, 0x1b512587cdb9749c,
0x57a4f7a2510253f5, 0xbe03d0d66bb8d368,
0xb504d6de6fbdd26b, 0x85feb35229644dd7,
0x4aadfdba5d0d50f0, 0xe063cf098a26ace9,
0x9684091c0e838d8a, 0x4d1aa591c679bfdc,
0x374d3da7ddad7090, 0x5ca3f1aa550752f6,
0x17e17ba452c89ab3, 0x8ef9b55a2d614cd4,
0x20ac46038f65ea23, 0x8411c4e673a6d562,
0x68c255cc66f197a4, 0xa80ddcc663b2d16e,
0xd099aa85ccff3355, 0x41aafbb2590851f3,
0x0f9cc7e2712a5bed, 0xae55f359a204a6f7,
0xc120febe5f81de7f, 0xa2e5ad7a3d7548d8,
0xcc7fd7299a32a8e5, 0x0ae871bc5ec799b6,
0xe63be0964b90db70, 0xdb9eac8dc8fa3256,
0x152295d1e651b7c4, 0xaace32b3d72bfc19,
0x7393704bab48e338, 0x3bfd638442dc9ebf,
0x52d041fc7eef91ae, 0x1ce67dac56cd9bb0,
0x78947643af4de23b, 0x6106bdb1d66dbbd0,
0xf1da9b32195841c3, 0xe5177957a5cb6eb2,
0xb35cf941ae0ba5f2, 0x564b80160bc0cb40,
0xc20c677fb1da6bbd, 0x7ecc59dc6efb95a2,
0x9f40e161be1fa1fe, 0xc3e310cbeb18f308,
0x2f3081e1fe4fb1ce, 0x160e0c10080a0206,
0x675e922e17dbcc49, 0x3f66a26e37f3c451,
0xcf534ee874691d27, 0x9c6c78a05044143c,
0x0e73b0562be8c358, 0x9a34573f91f263a5,
0xed3ce69e4f95da73, 0x358ed3d269345de7,
0x2380dfc2613e5fe1, 0xd72ef2ae578bdc79,
0x486e13cfe9947d87, 0x6c59942613decd4a,
0x5e601fdfe19e7f81, 0x049bc1ea752f5aee,
0xf3197547adc16cb4, 0x3e89d5da6d315ce4,
0xefff08ebfb0cf704, 0x47f2d42d98be266a,
0xb7c738abdb24ff1c, 0x11b9543b937eed2a,
0x36a24a13876fe825, 0x26f4699c4ed39dba,
0xee107f5fa1ce6fb1, 0x8b8d0304028c8e8f,
0xe34f56c8647d192b, 0x9447e769ba1aa0fd,
0xdeea1ad3e717f00d, 0xba98113c1e978986,
0x692d22783c330f11, 0x311512381c1b0709,
0xfd6ac5118629afec, 0x9bdb208bcb30fb10,
0x5838304020280818, 0x976b7ea85441153f,
0x7f232e6834390d17, 0x2c1c18201014040c,
0x0b07060804050103, 0xab2145078de964ac,
0xca27f8b65b84df7c, 0x0d5f2997c5b3769a,
0x64720beff980798b, 0xdc29f4a6538edd7a,
0xb2b38ef5f4c93d47, 0x8a6274b0584e163a,
0xa4bd82e5fcc33f41, 0xfc85b2a5dceb3759,
```

```
0xf81e734fa9c46db7, 0x95a890dde0d83848,
0x7708b1a1de67b9d6, 0x2a4437bfd1a27395,
0x3da54c1b836ae926, 0xea8bbeb5d4e1355f,
0x6db6e392491c55ff, 0x3c4a3bafd9a87193,
0x727c07fff18a7b8d, 0x9d830f140a868c89,
0x214331b7d5a77296, 0xb19f17341a928885,
0xe4f80ee3ff09f607, 0x33d6fc4da8822a7e,
0xafba84edf8c63e42, 0x2887d9ca653b5ee2,
0x4cf5d2259cbb2769, 0xc0cf890a054346ca,
0x74242860303c0c14, 0xa026430f89ec65af,
0xdf056d67bdd568b8, 0x8c3a5b2f99f861a3,
0x1d090a180c0f0305, 0x187dbc4623e2c15e,
0x7bb8ef82411657f9, 0x9918cefe7fa9d667,
0xf035ec86439ad976, 0x1295cdfa7d2558e8,
0xfb32ea8e479fd875, 0xbd2f491785e366aa,
0x921fc8f67bacd764, 0x83a69ccde8d23a4e,
0x4b428a0e07cfc845, 0xb9b488fdf0cc3c44,
0x90dc2683cf35fa13, 0x63c553c462f496a7,
0xa552f551a601a7f4, 0x01ef77b45ac298b5,
0x1abe5233977bec29, 0x7c0fb7a9da62b8d5,
0x226fa8763bfcc754, 0xf66dc319822caeef,
0xd4026b6fb9d069bb, 0xbfeca762317a4bdd,
0xd176dd31963dabe0, 0xc778d1219e37a9e6,
0xb6284f1f81e667a9, 0x4e363c5028220a1e,
0xcbc88f02014647c9, 0xc8e416c3ef1df20b,
0x032c99c1ee5bb5c2, 0x6beecc0d88aa2266,
0x4981647bb356e532, 0x0cb05e239f71ee2f,
0x461da399c27cbedf, 0x38d1fa45ac872b7d,
0xe2a0217c3ebf819e, 0xa67e6c90485a1236,
0xf4ae2d6c36b58398, 0xf5415ad86c771b2d,
0x622a247038360e12, 0x60e9ca058caf2365,
0xf9f104fbf306f502, 0xddc68312094c45cf,
0x76e7c61584a52163, 0x71509e3e1fd1ce4f,
0xa9e2ab72397049db, 0x09c4e87db09c2c74,
0x8dd52c9bc33af916, 0x54886e63bf59e637,
0x1e2593d9e254b6c7, 0x25d8f05da0882878,
0x816572b85c4b1739, 0xffa92b6432b0829b,
0xfe465cd068721a2e, 0xac961d2c169d8b80,
0xbcc03ea3df21fe1f, 0xa7911b2412988a83,
0x533f3648242d091b, 0x40458c0603cac946,
0xd8b2354c26a18794, 0x98f7b94a256b4ed2,
0x659d7c5ba342e13e, 0x1fcae46db8962e72,
0x42866273b753e431, 0x6e9a7a53a747e03d,
0x2bab400b8b60eb20, 0x59d747f47aea90ad,
0xb85bff49aa0ea4f1, 0xd25a44f078661e22,
0xcebc395c2eab8592, 0x873d5d279dfd60a0,
```

```
        0x0000000000000000, 0x5afbde3594b1256f,
        0xf2f602f3f703f401, 0xd5ed1cdbe312f10e,
        0x75cb5fd46afe94a1, 0x45313a582c270b1d,
        0x5f8f686bbb5ce734, 0x1056238fc9bc759f,
        0x07b7582b9b74ef2c, 0xe18cb8bdd0e4345c,
        0xc697a695c4f53153, 0x8f16c2ee77a3d461,
        0xa30adace67b7d06d, 0xd3b5334422a48697,
        0x556719d7e59b7e82, 0xeb64c9018e23adea,
        0xa1c934bbd32efd1a, 0x2edff655a48d297b,
        0xcd90a09dc0f03050, 0x88a19ac5ecd73b4d,
        0x30fa658c46d99fbc, 0x86d22a93c73ff815,
        0x2968ae7e3ff9c657, 0xad796a984c5f1335,
        0x3a121430181e060a, 0x271b1e281411050f,
        0x3461a46633f6c552, 0xbb77668844551133,
        0x06582f9fc1b67799, 0x436915c7ed917c84,
        0x797b01f7f58f7a8e, 0x6f750de7fd857888,
        0xf782b4add8ee365a, 0xc45448e0706c1c24,
        0x9eaf96d5e4dd394b, 0x1992cbf2792059eb,
        0xe84850c060781828, 0x70bfe98a451356fa,
        0x393e8df1f645b3c8, 0x243787e9fa4ab0cd,
        0x51fcd83d90b4246c, 0x7de0c01d80a02060,
        0x32398bf9f240b2cb, 0x4fd94be472e092ab,
        0x894eed71b615a3f8, 0x137aba4e27e7c05d,
        0xd6c1851a0d4944cc, 0x9133513795f762a6,
        0xb070608040501030, 0x082b9fc9ea5eb4c1,
        0xc5bb3f542aae8491, 0xe7d49722115243c5,
        0x44de4dec76e593a8, 0x0574b65e2fedc25b,
        0xb4eba16a357f4ade, 0x5b14a981ce73bdda,
        0x808a050c06898f8c, 0x02c3ee75b4992d77,
        0x5013af89ca76bcd9, 0x2df36f944ad69cb9,
        0xc90b6177b5df6abe, 0xfadd9d3a1d5d40c0,
        0x7a5798361bd4cf4c, 0x8249eb79b210a2fb,
        0xe9a727743aba809d, 0x93f0bf42216e4fd1,
        0xd95d42f87c631f21, 0x5d4c861e0fc5ca43,
        0xda71db399238aae3, 0xecd3912a155742c6,
};
const block T7[256] = {
        0x016ab9bb68d2d3ba, 0xb1669ae5194dfc54,
        0xcd1465e293bc712f, 0x511b8725b9cd9c74,
        0xa457a2f70251f553, 0x03bed6d0b86b68d3,
        0x04b5ded6bd6f6bd2, 0xfe8552b36429d74d,
        0xad4abafd0d5df050, 0x63e009cf268ae9ac,
        0x84961c09830e8a8d, 0x1a4d91a579c6dcbf,
        0x4d37a73daddd9070, 0xa35caaf10755f652,
        0xe117a47bc852b39a, 0xf98e5ab5612dd44c,
        0xac200346658f23ea, 0x1184e6c4a67362d5,
```

```
0xc268cc55f166a497 , 0x0da8c6dcb2636ed1 ,
0x99d085aaffcc5533 , 0xaa41b2fb0859f351 ,
0x9c0fe2c72a71ed5b , 0x55ae59f304a2f7a6 ,
0x20c1befe815f7fde , 0xe5a27aad753dd848 ,
0x7fcc29d7329ae5a8 , 0xe80abc71c75eb699 ,
0x3be696e0904b70db , 0x9edb8dacfac85632 ,
0x2215d19551e6c4b7 , 0xceaab3322bd719fc ,
0x93734b7048ab38e3 , 0xfd3b8463dc42bf9e ,
0xd052fc41ef7eae91 , 0xe61cac7dcd56b09b ,
0x947843764daf3be2 , 0x0661b1bd6dd6d0bb ,
0xdaf1329b5819c341 , 0x17e55779cba5b26e ,
0x5cb341f90baef2a5 , 0x4b561680c00b40cb ,
0x0cc27f67dab1bd6b , 0xcc7edc59fb6ea295 ,
0x409f61e11fbefea1 , 0xe3c3cb1018eb08f3 ,
0x302fe1814ffeceb1 , 0x0e16100c0a080602 ,
0x5e672e92db1749cc , 0x663f6ea2f33751c4 ,
0x53cfe84e6974271d , 0x6c9ca07844503c14 ,
0x730e56b0e82b58c3 , 0x349a3f57f291a563 ,
0x3ced9ee6954f73da , 0x8e35d2d33469e75d ,
0x8023c2df3e61e15f , 0x2ed7aef28b5779dc ,
0x6e48cf1394e9877d , 0x596c2694de134acd ,
0x605edf1f9ee1817f , 0x9b04eac12f75ee5a ,
0x19f34775c1adb46c , 0x893edad5316de45c ,
0xffefeb080cfb04f7 , 0xf2472dd4be986a26 ,
0xc7b7ab3824db1cff , 0xb9113b547e932aed ,
0xa236134a6f8725e8 , 0xf4269c69d34eba9d ,
0x10ee5f7fcea1b16f , 0x8d8b04038c028f8e ,
0x4fe3c8567d642b19 , 0x479469e71abafda0 ,
0xeaded31a17e70df0 , 0x98ba3c11971e8689 ,
0x2d697822333c110f , 0x153138121b1c0907 ,
0x6afd11c52986ecaf , 0xdb9b8b2030cb10fb ,
0x3858403028201808 , 0x6b97a87e41543f15 ,
0x237f682e3934170d , 0x1c2c201814100c04 ,
0x070b080605040301 , 0x21ab0745e98dac64 ,
0x27cab6f8845b7cdf , 0x5f0d9729b3c59a76 ,
0x7264ef0b80f98b79 , 0x29dca6f48e537add ,
0xb3b2f58ec9f4473d , 0x628ab0744e583a16 ,
0xbda4e582c3fc413f , 0x85fca5b2ebdc5937 ,
0x1ef84f73c4a9b76d , 0xa895dd90d8e04838 ,
0x0877a1b167ded6b9 , 0x442abf37a2d19573 ,
0xa53d1b4c6a8326e9 , 0x8beab5bee1d45f35 ,
0xb66d92e31c49ff55 , 0x4a3caf3ba8d99371 ,
0x7c72ff078af18d7b , 0x839d140f860a898c ,
0x4321b731a7d59672 , 0x9fb13417921a8588 ,
0xf8e4e30e09ff07f6 , 0xd6334dfc82a87e2a ,
0xbaafed84c6f8423e , 0x8728cad93b65e25e ,
```

```
0xf54c25d2bb9c6927, 0xcfc00a894305ca46,
0x247460283c30140c, 0x26a00f43ec89af65,
0x05df676dd5bdb868, 0x3a8c2f5bf899a361,
0x091d180a0f0c0503, 0x7d1846bce2235ec1,
0xb87b82ef1641f957, 0x1899fecea97f67d6,
0x35f086ec9a4376d9, 0x9512facd257de858,
0x32fb8eea9f4775d8, 0x2fbd1749e385aa66,
0x1f92f6c8ac7b64d7, 0xa683cd9cd2e84e3a,
0x424b0e8acf0745c8, 0xb4b9fd88ccf0443c,
0xdc90832635cf13fa, 0xc563c453f462a796,
0x52a551f501a6f4a7, 0xef01b477c25ab598,
0xbe1a33527b9729ec, 0x0f7ca9b762dad5b8,
0x6f2276a8fc3b54c7, 0x6df619c32c82efae,
0x02d46f6bd0b9bb69, 0xecbf62a77a31dd4b,
0x76d131dd3d96e0ab, 0x78c721d1379ee6a9,
0x28b61f4fe681a967, 0x364e503c22281e0a,
0xc8cb028f4601c947, 0xe4c8c3161def0bf2,
0x2c03c1995beec2b5, 0xee6b0dccaa886622,
0x81497b6456b332e5, 0xb00c235e719f2fee,
0x1d4699a37cc2dfbe, 0xd13845fa87ac7d2b,
0xa0e27c21bf3e9e81, 0x7ea6906c5a483612,
0xaef46c2db5369883, 0x41f5d85a776c2d1b,
0x2a6270243638120e, 0xe96005caaf8c6523,
0xf1f9fb0406f302f5, 0xc6dd12834c09cf45,
0xe77615c6a5846321, 0x50713e9ed11f4fce,
0xe2a972ab7039db49, 0xc4097de89cb0742c,
0xd58d9b2c3ac316f9, 0x8854636e59bf37e6,
0x251ed99354e2c7b6, 0xd8255df088a07828,
0x6581b8724b5c3917, 0xa9ff642bb0329b82,
0x46fed05c72682e1a, 0x96ac2c1d9d16808b,
0xc0bca33e21df1ffe, 0x91a7241b9812838a,
0x3f5348362d241b09, 0x4540068cca0346c9,
0xb2d84c35a1269487, 0xf7984ab96b25d24e,
0x9d655b7c42a33ee1, 0xca1f6de496b8722e,
0x8642736253b731e4, 0x9a6e537a47a73de0,
0xab2b0b40608b20eb, 0xd759f447ea7aad90,
0x5bb849ff0eaaf1a4, 0x5ad2f0446678221e,
0xbcce5c39ab2e9285, 0x3d87275dfd9da060,
0x0000000000000000, 0xfb5a35deb1946f25,
0xf6f2f30203f701f4, 0xedd5db1c12e30ef1,
0xcb75d45ffe6aa194, 0x3145583a272c1d0b,
0x8f5f6b685cbb34e7, 0x56108f23bcc99f75,
0xb7072b58749b2cef, 0x8ce1bdb8e4d05c34,
0x97c695a6f5c45331, 0x168feec2a37761d4,
0x0aa3cedab7676dd0, 0xb5d34433a4229786,
0x6755d7199be5827e, 0x64eb01c9238eeaad,
```

```
        0xc9a1bb342ed31afd, 0xdf2e55f68da47b29,
        0x90cd9da0f0c05030, 0xa188c59ad7ec4d3b,
        0xfa308c65d946bc9f, 0xd286932a3fc715f8,
        0x68297eaef93f57c6, 0x79ad986a5f4c3513,
        0x123a30141e180a06, 0x1b27281e11140f05,
        0x613466a4f63352c5, 0x77bb886655443311,
        0x58069f2fb6c19977, 0x6943c71591ed847c,
        0x7b79f7018ff58e7a, 0x756fe70d85fd8878,
        0x82f7adb4eed85a36, 0x54c4e0486c70241c,
        0xaf9ed596dde44b39, 0x9219f2cb2079eb59,
        0x48e8c05078602818, 0xbf708ae91345fa56,
        0x3e39f18d45f6c8b3, 0x3724e9874afacdb0,
        0xfc513dd8b4906c24, 0xe07d1dc0a0806020,
        0x3932f98b40f2cbb2, 0xd94fe44be072ab92,
        0x4e8971ed15b6f8a3, 0x7a134ebae7275dc0,
        0xc1d61a85490dcc44, 0x33913751f795a662,
        0x70b0806050403010, 0x2b08c99f5eeac1b4,
        0xbbc5543fae2a9184, 0xd4e722975211c543,
        0xde44ec4de576a893, 0x74055eb6ed2f5bc2,
        0xebb46aa17f35de4a, 0x145b81a973cedabd,
        0x8a800c0589068c8f, 0xc30275ee99b4772d,
        0x135089af76cad9bc, 0xf32d946fd64ab99c,
        0x0bc97761dfb5be6a, 0xddfa3a9d5d1dc040,
        0x577a3698d41b4ccf, 0x498279eb10b2fba2,
        0xa7e97427ba3a9d80, 0xf09342bf6e21d14f,
        0x5dd9f842637c211f, 0x4c5d1e86c50f43ca,
        0x71da39db3892e3aa, 0xd3ec2a915715c642,
};

#endif //CRYPTOGRAPHY_TTABLES_HPP
```