МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное образовательное учреждение
высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики, механики и компьютерных наук им. И. И. Воровича

Реферат

Криптоанализ шифра «Akelarre»

Выполнил: студент III курса, Радько В. А.
Проверил: доцент, Косолапов Ю. В.

Ростов-на-Дону, 2016

Содержание

1 Введение

«Akelarre» это блоковый шифр с секретным ключом. Главными операциями преобразования являются зависимые от данных циклические сдвиги, побитовое сложение и сложение в группе $\mathbb{Z}_{2^{32}}$. Операции образованы таким образом, что результат одной операции никогда не использовался как аргумент другой операции того же типа. Таким образом, любые отношения между сообщением, шифртекстом и ключом скрыты и сложны. Способ кодирования разработан так, чтобы быть уверенным, что каждый бит сообщения будет всегда учавствовать как минимум в одном сдвиге.

Результат каждой итерации, или каждого «раунда» – сложная функция, аргументом которой будет результат предыдущего «раунда» и нескольких под-блоков, основанных на секретном ключе пользователя.

Гибкость алгоритма шифрования основана на таких параметрах как длина ключа l и количество «раундов» r.

2 Алгебраическая модель шифра

Рассмотрим алгебраическую модель шифра:

$$A = (X, Y, K, f_k) \tag{1}$$

где $X = Y = \mathbb{Z}_2^{128}$, где X – пространство открытых текстов и Y – пространство шифртекстов. Открытый текст разбивается на блоки длины 128, если размер открытого текста нацело на 128 не делится, то текст дополняется произвольными символами. Ключевое пространство $K = \underbrace{\mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times ... \times \mathbb{Z}_2^{32}}_{13r+9}$ состоит из $13r+9$ 32-битных чисел $Z \in \mathbb{Z}_2^{32}$, каждое из которых будем называть «подключом» Z_i.

2.1 Описание алгоритма шифрования

Подробнее рассмотрим отображение $f_k(x) = y$. Считаем, что открытый текст был подготовлен к зашифрованию и разбит на 128-битовые блоки, поэтому можем рассматривать $x \in \mathbb{Z}_2^{128}$.

Алгоритм разбивается на три части: входная трансформация, r штук «раундов» и выходная трансформация. Входная трансформация требует 4 подключа, каждый раунд требует по 13 подключей, а выходная трансформация требует 5 подключей.

2.1.1 Входная трансформация

Во входной трансформации 128-битовый исходный текст разбивается на 4 блока по 32 бита, X_1, X_2, X_3, X_4. После этого X_1 и X_4 складываются в группе $\mathbb{Z}_{2^{32}}$ с $Z_1^{(0)}$ и $Z_4^{(0)}$, а X_2 и X_3 складываются побитово с $Z_2^{(0)}$ и $Z_3^{(0)}$ в $\mathbb{Z}_2$. Здесь и далее мы для

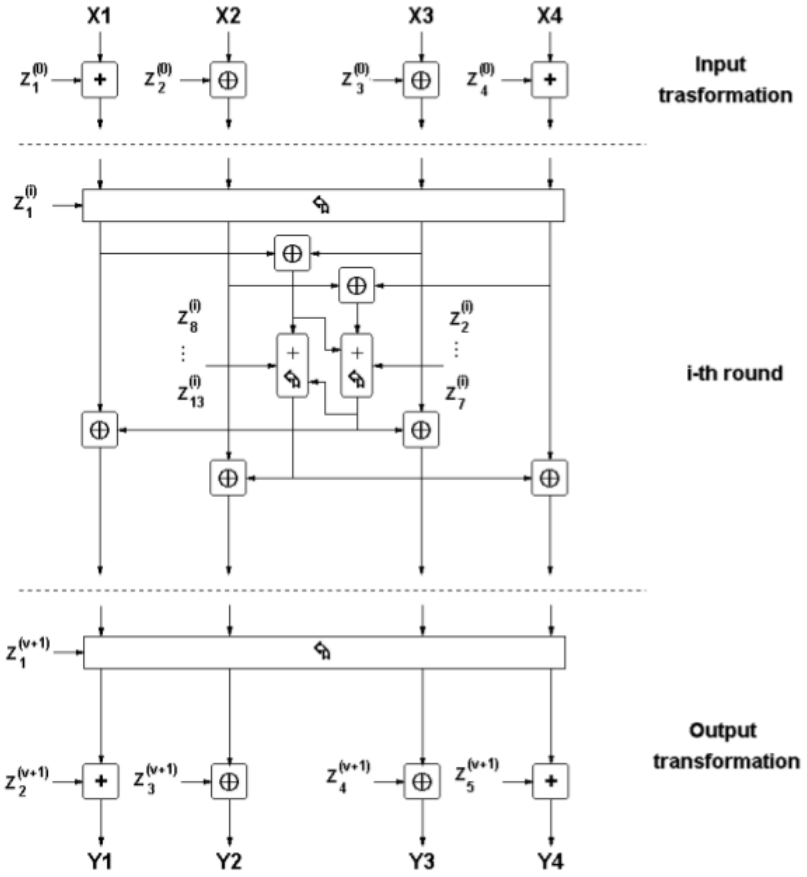

Рис. 1: Схема шифрования

удобства будем перенумеровывать подключи для каждой трансформации и каждого раунда, номер которой будем писать в верхнем индексе.

$$\begin{aligned} X_1' &= Z_1^{(0)} + X_1 \\ X_2' &= Z_2^{(0)} \oplus X_2 \\ X_3' &= Z_3^{(0)} \oplus X_3 \\ X_4' &= Z_4^{(0)} + X_4 \end{aligned} \tag{2}$$

После этой операции, $X_i, i = 1, ..., 4$ снова конкатенируются в 128-битный вектор.

2.1.2 Трансформация i-го раунда

Первой операцией в i-м раунде проводится циклический сдвиг 128-битного блока влево. Величина сдвига контролируется семью наименее значащими битами подключа $Z_1^{(i)}$. Затем получают временные числа $P_1 = X_1 \oplus X_3$ и $P_2 = X_2 \oplus X_4$, после чего они подаются на вход преобразованию, состоящему из сдвигов и сложений (Рис. 2).

В этом преобразовании P_2 выполняет циклические сдвиги в соответствии с P_1, при этом после каждого сдвига к P_2 прибавляются ключи $Z_j^{(i)}$. Преобразования P_1 зависят от Q_2, полученном из P_2. Сдвиги выполняются над блоками длины 31 бит,

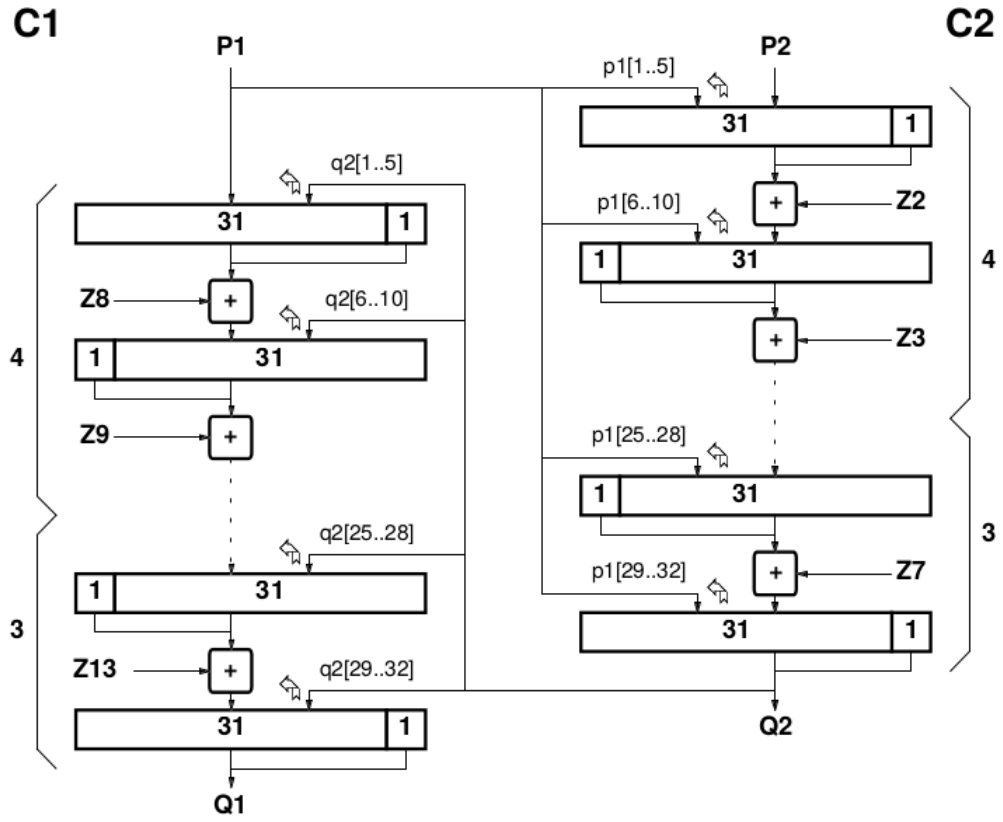

Рис. 2: Преобразование, состоящее из сдвигов и сложений

при этом один из крайних битов (правый или левый) не изменяются поочерёдно. При этом первые четыре сдвига P_2 зависят от выбранных пяти бит P_1, а последние три – от выбранных четырёх бит. Аналогично и для P_1.

После завершения вычисления Q_1 и Q_2, результат конкатенируется из следующих блоков:

$$\begin{aligned} X'_1 &= X_1 \oplus Q_2 \\ X'_2 &= X_2 \oplus Q_1 \\ X'_3 &= X_3 \oplus Q_2 \\ X'_4 &= X_4 \oplus Q_1 \end{aligned} \tag{3}$$

2.1.3 Выходная трансформация

После того, как были проведены r раундов, проходит завершающая трансформация. Первое, что она делает – циклический сдвиг влево 128-битового блока, его порядок зависит от семи наименее значащих битов подключа $Z_1^{(r+1)}$. Следующие трансформации аналогичны входным трансформациям:

$$
\begin{aligned}
X_1' &= Z_2^{(r+1)} + X_1 \\
X_2' &= Z_3^{(r+1)} \oplus X_2 \\
X_3' &= Z_4^{(r+1)} \oplus X_3 \\
X_4' &= Z_5^{(r+1)} + X_4
\end{aligned}
\tag{4}
$$

Полная схема шифрования на рисунке 1.

2.2 Алгоритм расшифрования

Алгоритм расшифрования аналогичен алгоритму зашифрования, единственное отличие в том, что ключи расшифрования вычисляются из ключей для зашифрования представленным в таблице 1 образом.

Таблица 1: Ключи для расшифрования

Раунд	Ключи шифрования	Ключи расшифрования
Входная трансформация (0)	$Z_1^{(0)}, Z_2^{(0)}, Z_3^{(0)}, Z_4^{(0)}$	$-Z_2^{(r+1)}, Z_3^{(r+1)}, Z_4^{(r+1)}, -Z_5^{(r+1)}$
1	$Z_1^{(1)}, ..., Z_{13}^{(1)}$	$\left(Z_1^{(r+1)}\right)^{-1}, Z_2^{(r)}, ..., Z_{13}^{(r)}$
2	$Z_1^{(2)}, ..., Z_{13}^{(2)}$	$\left(Z_1^{(r)}\right)^{-1}, Z_2^{(r-1)}, ..., Z_{13}^{(r-1)}$
...	...	...
i	$Z_1^{(i)}, ..., Z_{13}^{(i)}$	$\left(Z_1^{(r+2-i)}\right)^{-1}, Z_2^{(r+1-i)}, ..., Z_{13}^{(r+1-i)}$
...	...	...
r	$Z_1^{(r)}, ..., Z_{13}^{(r)}$	$\left(Z_1^{(2)}\right)^{-1}, Z_2^{(1)}, ..., Z_{13}^{(1)}$
Выходная трансформация $(r+1)$	$Z_1^{(r+1)}, Z_2^{(r+1)}, Z_3^{(r+1)}, Z_4^{(r+1)}, Z_5^{(r+1)}$	$\left(Z_1^{(1)}\right)^{-1}, -Z_1^{(0)}, Z_2^{(0)}, Z_3^{(0)}, -Z_4^{(0)}$

2.3 Алгоритм генерации ключевого потока

Поскольку алгоритм требует $13r+9$ ключей, неразумно было бы требовать все их от пользователя. Следовательно необходимо получив некоторое начальное значение, мастер-ключ, генерировать ключевой поток.

Схема алгоритма генерации ключевого потока представлена на рисунке 3. Пользователь может выбирать ключ любой длины. Он будет разделен на m частей k_i, каждая по 16 бит. Каждая часть возводится в квадрат и к ней прибавляется константа A_0 или A_1 по модулю 2^{32}, согласно схеме. Главная причина, по которой к ключу прибавляются константы в том, чтобы избежать нежелательного эффекта, когда пользователь ввёл нулевой ключ. Константы должны выбираться так, чтобы $A_0, A_1 \neq 2^{32} - n^2, n \in \mathbb{Z}$, и с хорошим статистическим распределением нулей и еди-

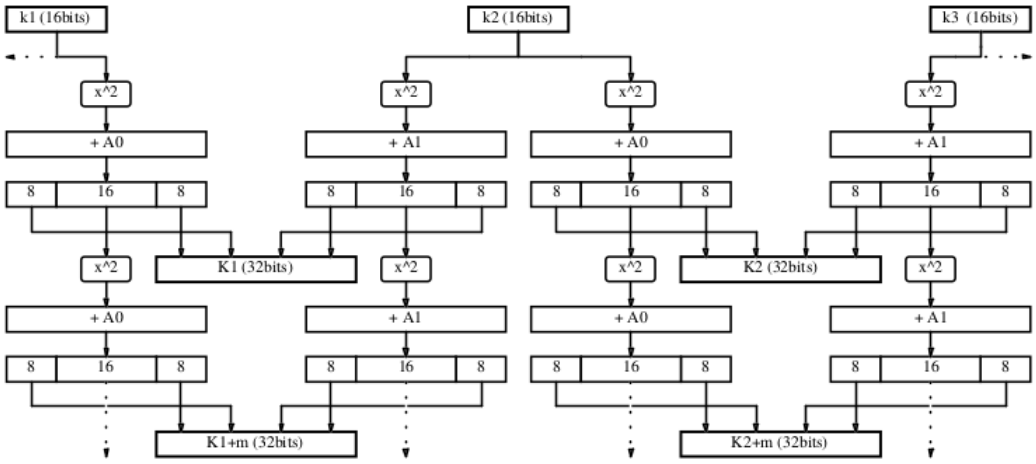

Рис. 3: Схема ключевого потока

ниц, потому что результат возведения в квадрат будет 2^{32}, что будет представлено тридцатью двумя нулями. Авторы статьи предлагают следующие значения [1]:

$$\begin{aligned} A_0 &= a49ed284_{16} \\ A_1 &= 735203de_{16} \end{aligned} \tag{5}$$

После того, как эти значения прибавлены, внутренние 16 бит каждой пары передаются в следующую итерацию генерации ключевого потока, а четыре оставшихся блока по 8 бит, конкатенируются указанном в схеме образом в 32-битный подключ.

3 Криптоанализ [2]

Основное наблюдение, на котором строится криптоанализ, состоит в том что преобразование одного раунда сохраняет чётность входного сообщения. Циклический сдвиг 128 битового блока не влияет на чётность, а подпоследовательности структуры сложений и сдвигов производит сложение (mod 2) дважды, следовательно сохраняет чётность. Единственные операции в алгоритме шифрования Akelarre, которые влияют на чётность, это входная трансформация и выходная трансформация. Это позволяет нам атаковать блоки ключей, задействованные в этих трансформациях независимо от фукнции преобразования раундов.

Мы реализуем атаку на выбранный текст в 4 фазы. В первой фазе мы находим большинство битов из двух подключей выходной трансформации. Во второй фазе находим большинство битов двух подключей входной трансформации. В третьей фазе используем последовательность ключей, чтобы восстановить 80 бит информации о мастер-ключе. В четвёртой фазе ищем мастер-ключ среди все оставшихся возможных ключей полным перебором.

3.1 Восстановление битов подключа выходной трансформации

Мы начинаем с фиксирования $X_1 = 0$ и $X_4 = 0$, и зашифровываем множество блоков со случайными значениями X_2 и X_3. Обозначим $\mathcal{P}(\cdot, \cdot, ...)$ чётностью конкатенации всех своих аргументов (сумма всех битов (mod 2)).

Определим:

$$\begin{aligned} k &= \mathcal{P}(Z_1^{(0)}, Z_2^{(0)}, Z_3^{(0)}, Z_4^{(0)}) \\ x &= \mathcal{P}(X_2, X_3) \\ r &= \mathcal{P}(R_1^{(0)}, ..., R_4^{(0)}) \end{aligned} \tag{6}$$

легко видеть, что $r = k \oplus x$.

Поскольку функция преобразования раунда инвариантна относительно чётности, мы имеем $r = \mathcal{P}(R_1^{(v)}, ..., R_4^{(v)})$ после v раундов, и следовательно $r = \mathcal{P}(S_1^{(v+1)}, ..., S_4^{(v+1)})$.

Пусть $K_1 = -Z_2^{(v+1)} \mod 2^{32}$, и $K_4 = -Z_5^{(v+1)} \mod 2^{32}$. Это даёт нам

$$r = \mathcal{P}\left((Y_1 + K_1) \mod 2^{32}, Y_2 \oplus Z_3^{(v+1)}, Y_3 \oplus Z_4^{(v+1)}, (Y_4 + K_4) \mod 2^{32}\right).$$

Собирая все формулы мы получаем:

$$\mathcal{P}\left((Y_1 + K_1) \mod 2^{32}, (Y_4 + K_4) \mod 2^{32}\right) = k' \oplus x \oplus y. \tag{7}$$

где $k' = k \oplus \mathcal{P}\left(Z_3^{(v+1)}, Z_4^{(v+1)}\right)$ и $y = \mathcal{P}\left(Y_2, Y_3\right)$. Определим $\forall K, K^* = K[30..0]$ число, сформированное наименее значимыми 31 битами K. Разбивая наиболее значимые биты суммы, мы можем переписать уравнение (7) в виде

$$\mathcal{P}\left(Y_1^* + K_1^*, Y_4^* + K_4^*\right) = k'' \oplus x \oplus y' \tag{8}$$

где $k'' = k' \oplus K_1[31] \oplus K_4[31]$ и $y' = y \oplus Y_1[31] \oplus Y_4[31]$. Значение k'' зависит только от ключа и будет одинаковым для всех наших шифрований. Значения x и y' известны, поскольку зависят только от открытого текста или шифртекста.

Если мы найдём два шифртекста i и j с одинаковыми значениями Y_1^* ($Y_{1,i}^* = Y_{1,j}^*$), тогда мы можем получить отношение для K_4^*. Мы получим

$$\mathcal{P}\left(Y_{4,i}^* + K_4^*\right) \oplus \mathcal{P}\left(Y_{4,j}^* + K_4^*\right) = x_i \oplus x_j \oplus y_i' \oplus y_j' \tag{9}$$

Это уравнение исключает около половины возможных значений K_4^*. После $4 \cdot 10^5$ выбранных открытых текстов, вы можем ожидать около 37 отдельных совпадений Y_1^*, и следовательно около 37 отношений (9) для K_4^*. Теперь мы можем найти перебором из 2^{31} возможных значений K_4^* такое значение, которое будет удовлетворять всем отношениям чётности. Численные эксперименты показывают, что 37 отноше-

ний обычно хватает, чтобы получить единственное решение. Когда K_4^* будет найден, каждое полученное зашифрование даёт эквивалентные отношения для K_1^*, которые позволяют нам так же перебором найти K_1^*.

В итоге, эта фаза атаки требует около $4 \cdot 10^5$ выбранных открытых текстов, и 2^{32} шагов перебора, чтобы восстановить и K_1^*, и K_4^*. Этот перебор можно ускорить. Алгоритм генерирования ключей не может сгенерировать все 2^{32} возможных подключей, эта информация может быть использована для ускорения переборного алгоритма. Как очевидно из определения генератора ключевого потока, все возможные подключи могут быть пронумерованы перечислением возможных значений двух половин подключей по-отдельности. Эти сужает количество наименее значимых 31 бит подключей выходной трансформации до около 2^{25} возможных значений (при условии, что количество раундов 4, и учитывая природу ключевой последовательности, в которой энтропия подключей выходной трансформации уменьшается при увеличении количества раундов).

Последняя фаза нашей атаки это полный перебор, который требует полное зашифрование каждым мастер-ключом из около 2^{48} возможных. Проверка 2^{50} возможных ключевых значений используя отношения чётности определённо потребует меньше работы. Это ведёт к следующему улучшению: используя только 60 выбранных открытых текстов, мы ищем K_1^* и K_4^* параллельно, используя уравнение (8). Существует около 2^{25} возможных значений для каждого из этих двух значений, которые дают нам в сумме 2^{50} возможных значений для пары. Мы можем ожидать найти правильные значения (которые удовлетворяют всем отношениям чётности) приблизительно за 2^49 попыток. Вычислительные мощности необходимые в этой фазе всё ещё незначительны, по сравнению с необходимыми мощностями в последней фазе атаки, поскольку каждая операция в этой фазе гораздо менее сложная.

Перебор может быть улучшен ещё сильнее, если мы примем во внимание неравномерность распределения блоков ключей. Из определения генератора ключевого потока можно легко получить распределение вероятностей всех 2^{25} возможных подключей, это может быть сделано вычислением независимых вероятностей для каждой из двух половин подключей. Наши результаты показывают, что это оставляет около 23.5 бит энтропии для каждого значения K^*. Посредством поиска значений у которых высокая вероятность выше, мы можем получить корректные значения ключа гораздо быстрее.

3.2 Восстановление битов подключа входной трансформации

Мы так же можем восстановить 31 наименее значимых бита $Z_1^{(0)}$ и $Z_4^{(0)}$. Можно, конечно, проанализировать все предшествующие секции функции расшифрования, но существуют более непосредственные методы.

Как только мы получили K_1^* и K_4^*, мы можем распознать какая из двух шифровок имеет одинаковую чётность в течение раундов (мы можем расшифровать достаточно

выходной трансформации; ключевые биты, которые мы не знаем, влияют на чётность одним и тем же образом при каждом зашифровании). Выберем фиксированные X_1, X_2 и X_3, и проведём зашифрования для разных значений X_4. Это даёт нам отношения чётности для $Z_4^{(0)*}$ аналогичные уравнению (9). Используя те же методы, как и для предыдущих шагов, мы можем следовательно восстановить 31 наименее значащих бит $Z_4^{(0)}$ и $Z_1^{(0)}$, используя 2^{32} прямых переборных шага и около 80 выбранных открытых текстов.

Более непостредственный метод так же возможен, где каждое зашифрование выбранного открытого текста открывает один бит в $Z_4^{(0)*}$ или $Z_1^{(0)*}$. Это уничтожает переборный поиск этих 31 битных значений, и уменьшает количество выбранных открытых текстов на этой фазе до 62.

3.3 Восстановление информации о мастер-ключе из подключей

Мы получили 31 наименее значащих бита четырёх подключей. Из структуры генератора ключевого потока, каждая половина подключа зависит от 16 бит мастер-ключа.

Таблица 2 даёт ожидаемую информацию, предоставленную частично известными подключами о блоках мастер ключа, предполагая, что мастер-ключ выбран случайным образом равновероятно.

Таблица 2: Биты информации, предоставленные подключом a мастер-ключе

Подключ	**«верхняя часть»**	**«нижняя часть»**
$Z_1^{(0)*}$	11.99 бит о k_1	12.85 бит о k_2
$Z_4^{(0)*}$	11.99 бит о k_4	12.85 бит о k_5
$Z_2^{(5)*}$	11.52 бит о k_2	12.01 бит о k_3
$Z_5^{(5)*}$	11.52 бит о k_5	12.01 бит о k_6

Отображение из блока мастер-ключа в одну из половин подключа, не биекция, не все 2^16 возможных значений подключа могут возникнуть. Фактически, каждый 32-битный подключ имеет от 24.1 до 25.7 битов энтропии. Некоторые из блоков мастер-ключа влияют на два восстановленных подключа. В этом случае мы можем ожидать, что у нас останется единственно возможное значение блока мастер-ключа (так как существует только 16 бит в блоке мастер ключа, мы не можем иметь более чем 16 бит информации о нём). Интересное наблюдение в том, что количество информации, которое мы получаем о мастер-ключе зависит хаотично от количества раундов, из-за вырванивания известных подключей ключевого потока. В некоторых случаях известные подключи все произведены из четырёх блоков мастер-ключа, в то время как в других случаях они получены из семи блоков мастер-ключа. Если мы увеличим число раундов до 5, мы можем получить на 7 бит информации о блоках мастер ключа больше. Делая 5 раундов, Akelarre становится значительно уязвимее для нашей

атаки, чем 4-х раундовая версия.

3.4 Восстановление полного мастер-ключа

Собрав информацию, которую мы получили, мы можем получить 80 бит информации о 128-битном ключе. Это оставляет нам около 2^{48} возможных значений мастер-ключа. Их легко перенумеровать: для каждого блока мастер-ключа мы создадим список всех возможных значений. Для тех блоков мастер-ключей, которые влияют на известные подключи, мы пробуем все 2^{16} возможных значений и отбрасываем те, которые не удовлетворяют известным битам подключа. После этого мы будем иметь два блока мастер-ключа полностью известные, 4 блока мастер-ключа, которые известны частично и два блока мастер-ключа, которые полностью неизвестны. Декартово произведение этих восьми списков перечисляет возможные значения мастер-ключа. Используя прямой перебор через эти возможные значения, мы можем найти полный 128-битный мастер-ключ через, как максимум, 2^{48} попыток.

4 Реализация алгоритма шифрования

Поскольку алгоритм подразумевает множество битовых операций и сдвигов, было решено использовать язык C, поскольку он достаточно низкоуровневый и позволяет организовывать с помощью **union** доступ к одному и тому же участку памяти, разных по виду, но одинаковых по размерам структур.

Программа состоит из трёх модулей (единиц компиляции): «akelarre», «keygen» и главного модуля «main».

В модуле «keygen» описаны две функции:

```
uint32_t get_next_key(uint32_t* k);
void gen_decrypt_keys(uint32_t* k, uint32_t* d, uint32_t nrounds);
```

Первая генерирует следующий ключ в цепочке по предыдущему. Программа ограничивает вариативность теоретического алгоритма генерации ключа входной последовательностью в 32 бита. Вторая функция по заданному массиву ключей и количеству раундов генерирует ключи расшифрования, согласно схеме в таблице 1.

Модуль «akelarre» содержит алгоритм шифрования, а так же вспомогательные функции циклических сдвигов, требуемых алгоритму: циклического сдвига старших и младших 31 бит в 32-х битном слове, а так же сдвига 128 битного слова.

Модуль «main» содержит в себе алгоритм, который по полученным аргументам командной строки считает текст из файла, зашифрует (расшифрует) и запишет шифртекст в указанный файл, а так же запишет в указанный файл ключи, которыми происходило зашифрование. Инициализацию ключевого потока программа производить самостоятельно не позволяет, инициализация происходит случайным 32-битным значением. Последний аргумент r – количество раундов, вариативный, по-

умолчанию используется 1 раунд, но пользователь может указать своё количество. Использование программы:

```
./akelarre {encrypt|decrypt} input_file key_file output_file [r]
```

Полный исходный код можно посмотреть в репозитории Github [3]

Список использованных источников

1. Akelarre: a new Block Cipher Algorithm / G. Álvarez [и др.]. — 1996.
2. *Ferguson N.*, *Schneier B.* Cryptanalysis of Akelarre. — 1997.
3. Репозиторий с исходными кодами. — URL: `https://github.com/justslavic/akelarre_cipher` (дата обр. 30.12.2017).