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 $X = Y = \mathbb{Z}_2^n$, где X – пространство открытых текстов и Y – пространство шифртекстов. Открытый текст $x = (x_1, x_2, ..., x_n)$ разбивается на блоки длины 128, если 128 не делит n нацело, то вектор x дополняется произвольными (возможно нулевыми) значениями. Ключевое пространство $K = \underbrace{\mathbb{Z}_2^{32} \times \mathbb{Z}_2^{32} \times ... \times \mathbb{Z}_2^{32}}_{13r+9}$ состоит из $13r + 9$ 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

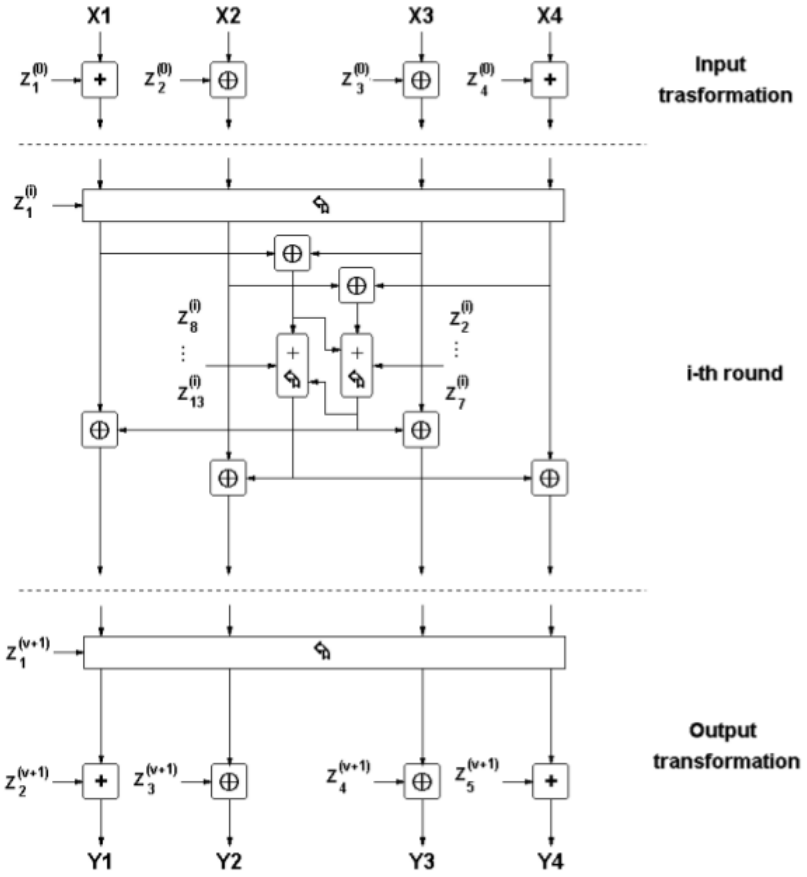

Рис. 1: Схема шифрования

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 i-й раунд

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. 2.1.2).

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,

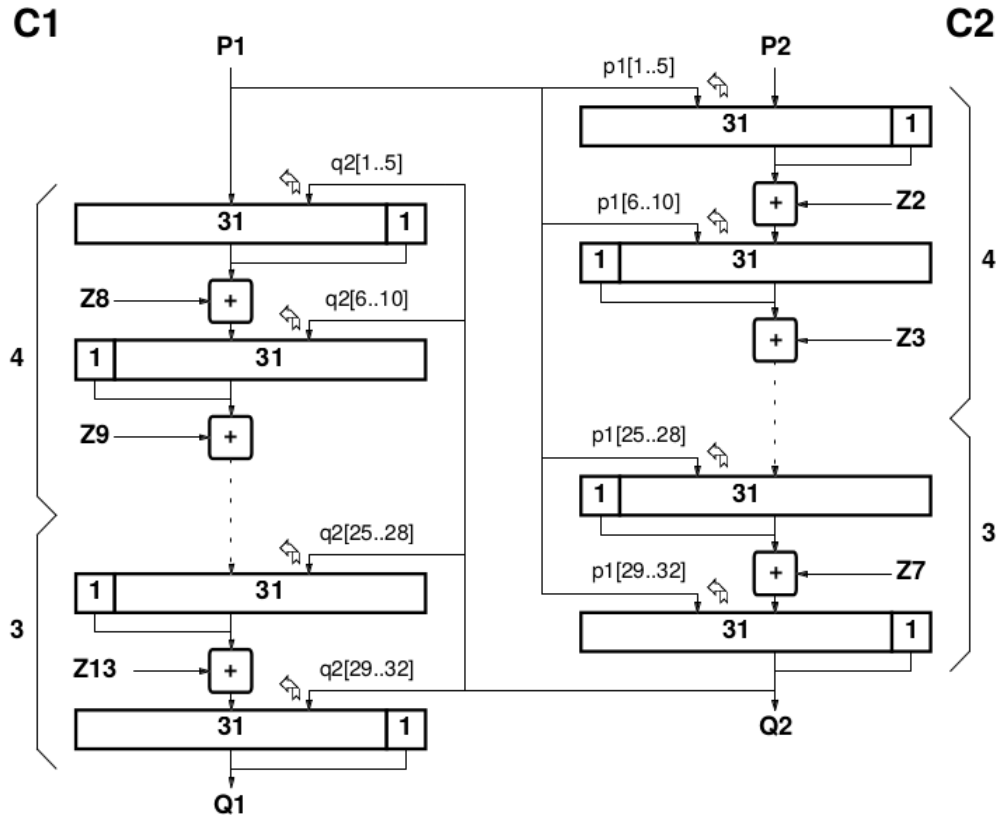

Рис. 2: Преобразование, состоящее из сдвигов и сложений

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 2.1.

2.2 Алгоритм расшифрования

Алгоритм расшифрования аналогичен алгоритму зашифрования, единственное отличие в том, что ключи расшифрования вычисляются из ключей для зашифрования образом, представленным в таблице 2.2:

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, генерировать ключевой поток.

Схема алгоритма генерации ключевого потока представлена на рисунке 2.3. 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-

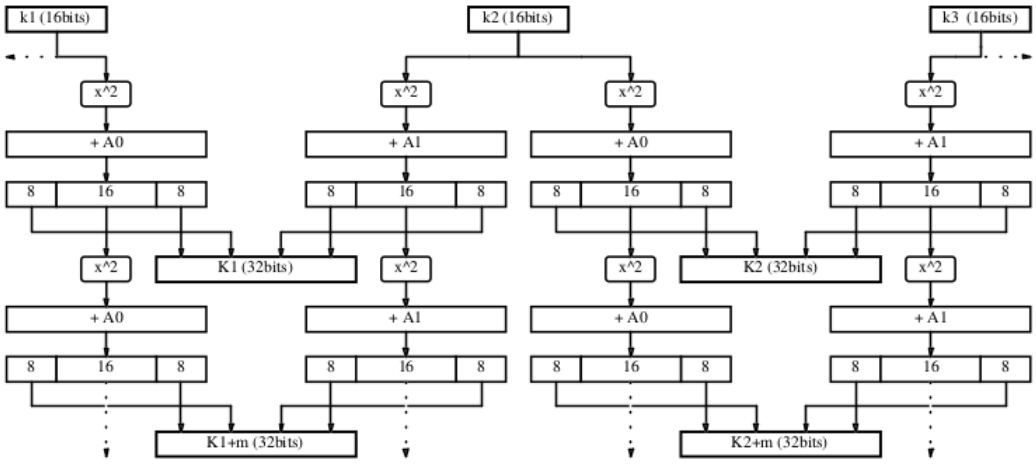

Рис. 3: Схема ключевого потока

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 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 2.2.

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).