МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Механико-математический факультет

Кафедра математической теории интеллектуальных систем

Курсовая работа

АППАРАТНАЯ РЕАЛИЗАЦИЯ НИЗКОРЕСУРСНОГО БЛОЧНОГО ШИФРА GAGE-InGAGE, ОСНОВАННОГО НА КВАЗИГРУППАХ HARDWARE IMPLEMENTATION OF THE LIGHTWEIGHT QUASIGROUP BLOCK CIPHER 'GAGE-InGAGE'

Подготовила:
Студентка 305 группы
Елисеева Анастасия Васильевна

Научный руководитель:
Галатенко Алексей Владимирович

Москва, 2024г.

1 Аннотация

В курсовой работе рассматривается низкоресурсный криптографический алгоритм GAGE-InGAGE [1] с нелинейным слоем, основанном на действии квазигруппы. Приведены определения и утверждения из квазигрупповой теории, фигурирующие в теоретическом обосновании работы алгоритма: d- и e-преобразования и их свойства. Описаны принципы функционирования некоторых криптографических структур: регистровых сдвигов с линейной обратной связью, блочных шифров с губчатой структурой; рассмотрены характеристики сложности и глубины схем, а также приведены некоторые связанные утверждения. Создана оптимизированная по сложности схемная реализация алгоритма, описанная на языке Verilog.

2 Введение

2.1 Актуальность задач квазигрупповой и низкоресурсной криптографии

Современные технологии должны отвечать запросам пользователей на улучшение качества жизни с помощью небольших электронных девайсов: на рынке появляется все больше устройств, анализирующих биологические данные пользователей, автоматизирующих их быт или передающих данные более удобным для пользователя образом. Из-за расширения применения микроконтроллеров вплоть до медицинских, финансовых, военных структур и других сфер, оперирующих с конфиденциальными данными, все актуальнее становятся криптографические алгоритмы, обладающие достаточной надежностью при легковесной аппаратной реализации. Для создания таких алгоритмов можно использовать квазигруппы: благодаря существованию методов генерации квазигрупп и возможности применения алгебраического аппарата структуры на их основе удобны для описания, анализа и исследования, а свойства квазигрупп делают их применимыми для создания S-блоков с нужными свойствами.

2.2 Конкурс NIST и алгоритм GAGE-InGAGE

В 2013 Национальный Институт Стандартов и Технологий США (NIST) создал проект [2], посвященный развитию низкоресурсной криптографии, призванный изучить показатели безопасности известных криптографических алгоритмов, применяемых в устройствах с ограниченными ресурсами, чтобы разработать стандарт для этого направления криптографии. В 2017 году было принято решение привлечь к созданию портфеля алгоритмов исследовательские группы вне института. Так, к февралю 2023 был проведен конкурс в несколько раундов, участие в котором приняла в том числе команда разработчиков GAGE-InGAGE (Danilo Gligoroski, Norway; Hristina Mihajloska, Macedonia; Daniel Otte, Germany; Mohamed El-Hadedy, USA). Хоть алгоритм не имел особого успеха в конкурсе, его структура вызывает интерес из-за возможности предметного исследования использования квазигрупп в сочетании с более распространенными в криптографии объектами.

2.3 Структура работы

В начале будут разобраны основные связанные понятия алгебры квазигрупп, теории минимизации ДНФ и синтеза схем, затем описана структура алгоритма GAGE-InGAGE, показаны синтезированные схемы и описаны их характеристики.

3 Основные определения и обозначения

3.1 Квазигруппы [3] и перестановки

Квазигруппа (Q, *) – это группоид такой, что $\forall u, v \in Q \quad \exists! x, y \in Q : \quad u{*}x = v, \quad y{*}u = v$. Квазигруппы можно представлять в виде таблицы умножения *, причем полученная таблица оказывается *латинским квадратом*, то есть каждый элемент Q встречается в каждых строке и столбце ровно 1 раз.

Бинарная операция * индуцирует бинарное *левое сопряжение*: $x := a\backslash_* b \iff a * x = b$
Тогда (Q, $\backslash_*$) – тоже квазигруппа, и в алгебре (Q, *, $\backslash_*$) выполнются соотношения
$x\backslash_*(x * y) = y = x * (x\backslash_* y)$.

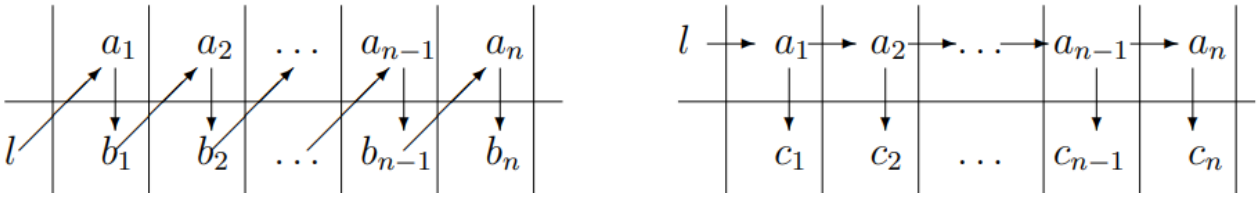
Обозначим за Q$^+$ множество непустых *слов* (конечных последовательностей элементов) вида $M = a_0 a_1 ... a_n = (a_0, a_1, ..., a_n)$ в алфавите (конечном множестве) Q.
На этом множестве можно ввести *e-преобразования* и *d-преобразования* $Q^+ \to Q^+$, определенные так:

$$e(l,*)(M) := b_1 b_2 ... b_n, \quad b_1 = l * a_1, \quad b_2 = b_1 * a_2, \quad ..., \quad b_n = b_{n-1} * a_n$$

$$d(l,*)(M) := c_1 c_2 ... c_n, \quad c_1 = l * c_1, \quad c_2 = a_1 * a_2, \quad ..., \quad c_n = a_{n-1} * a_n$$

l здесь – *лидерная константа* или *лидер.*

Графические представления e-приобразования и d-преобразования:



Вычисление d-преобразования можно производить параллельно, в то время как каждый символ результата e-преобразования, кроме первого, зависит от предыдущего. В основе использования этих преобразований для шифрования сообщений лежит следующая связь:

Теорема[3]. *$e_{l,*}$ и $d_{l,\backslash_*}$ – взаимно обратные преобразования.*

Пусть $S = (s_0, ..., s_{b-1})$ – состояние, π_S – перестановка, $b \in \mathbb{N}$ – четное.

$Paring(S) = S_{pairs} := \{\{s_0, s_1\}, \{s_2, s_3\}, ..., \{s_{b-2}, s_{b-1}\}\}$ – разбиение S на последовательные пары.
Скажем, что у π_S есть *сохраняющий пары период* t, если $\exists\{s_{2v}, s_{2v+1}\} \in Pairing(\pi^t(S))$.
Максимальная сохраняющая пары перестановка [1] – та, для которой сохраняющий пары период $t = \frac{b}{2}$.

3.2 Минимизация ДНФ [4]

Элементарная конъюнкция – выражение вида $x_{i_1}^{\sigma_1} \& x_{i_2}^{\sigma_2} \& ... \& x_{i_k}^{\sigma_k}$,
где $x^\sigma = \begin{cases} x, & \sigma = 1 \\ \bar{x}, & \sigma = 0 \end{cases}$ и $i_s < i_t$ для $s < t$.
Тогда *дизъюнктивная нормальная форма* (ДНФ) $D = U_1 \vee U_2 \vee ... \vee U_k$, где U_i – попарно различные элементарные конъюнкции.
На всевозможных ДНФ функции вводится мера сложности:
$L(x_{i_1}^{\sigma_1} \& x_{i_2}^{\sigma_2} \& ... \& x_{i_k}^{\sigma_k}) = k(\pm 1)$
$L(D) = \sum_{i=1}^{k} L(U_i)(\pm k)$.
Минимальная ДНФ (minДНФ) – та ДНФ функции, мера сложности которой минимальна (их может быть несколько).

Для минимизации ДНФ удобно использовать геометрический подход: отождествим функцию $f \in \bar{P}_2^n$ (множество булевых функций от n переменных, не считая константы 0 и 1) с подмножеством точек на E_2^n (булевом кубе), в которых функция принимает значение 1.
Если N_{K_i} – множество единиц конъюнкции K_i, то $N_{K_1} \cup N_{K_2} \cup ... \cup N_{K_m} = N_f$, то есть построение ДНФ эквивалентно покрытию подмножества вершин гранями.
Грань, соответствующую простой иппликанте, назовем *максимальной.*
$J_p^m(f)$ – множество таких импликант f.
$\bigvee_{K \in J_p^m(f)}$ - *сокращенная* ДНФ.
Тупиковая ДНФ (ТДНФ) – подмножество сокДНФ функции, в котором никакая импликанта не является избыточной. minДНФ - одна из тупиковых. Их множество - $T(f)$.

Утверждение. *minДНФ получается из сокДНФ путем отбрасывания некоторого (возможно пустого) количества слагаемых.*

α – ядровый набор для f, если $\exists! K \in J_p^m(f) : K(\alpha) = 1$. Этой конъюнкции соответствует *ядровая* грань. Множество всех ядровых граней образует *ядро* $J_a^n(f)$.

Утверждение. $K \in J_a^n(f) \iff K \in \bigcap T(f)$.

Утверждение. *ДНФ, полученная отбрасыванием из сокДНФ всех не ядровых простых импликант (ДНФ Квайна) и, возможно, какого-то еще количества слагаемых, является минимальной.*

С доказательствами этих утверждений можно ознакомиться в [4].

3.3 Автоматы [5]

(*Конечный абстрактный детерминированный*) *автомат* – объект $V = (A, Q, B, \phi, \psi)$, где A, Q, B – *входной алфавит, алфавит состояний, выходной алфавит* соответственно (конечные множества), $\phi : Q \times A \to Q$ – *функция переходов*, $\psi : Q \times A \to B$ – *функция выходов*.
Если функция $\psi(q, a)$ зависит от второго аргумента фиктивно, автомат называется *автоматом Мура*.

3.4 Схемы и их характеристики [4]

Схема из функиональных элементов (СФЭ) определена несколькими компонентами:

- Элементарный *базис* функциональных элементов (ФЭ) с n_i входами и 1 выходом: $\{F_i\}_1^r$;
- Логическая сеть, определенная индуктивно. Изолированная вершина представляет собой провод, а ряд операций позволяет сделать индуктивный переход:
  - объединение схем;
  - присоединение функционального элемента входами к выходам схемы;
  - разветвление выходов для подключения к нескольким входам (и склейка входов).

Функционирование такой схемы определено индуктивно. Фактически, СФЭ реализует булев оператор $\begin{pmatrix} f_1 \\ \vdots \\ f_m \end{pmatrix} (x_1, ..., x_n) = \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix}$.
Соединение данных входов, выходов и пронумерованных ФЭ – функция такая, что:

- любой вход ФЭ подключен либо к входу соединения, либо к выходу ФЭ;
- любой выход соединения подключен либо к входу соединения, либо к выходу ФЭ.

Для теоретической оценки скорости работы и компактности аппаратного исполнения схемы из функциональных элементов вводятся характеристики L-сложности и D-глубины соответственно:
$L(C)$ = число ФЭ (с весами) схемы C,
$D(C)$ = число ФЭ на самом длинном «пути» от входа к выходу в схеме C.
Также для дальнейших оценок можно ввести функции Шеннона глубины и сложности:
$L(f) = \min\{L(C) : C \sim f\}, \quad L(n) = \max\{L(f) : f \in P_2^n\}$,
$D(f) = \min\{D(C) : C \sim f\}, \quad D(n) = \max\{D(f) : f \in P_2^n\}$, где P_2^n - все булевы функции от n переменных, C - схема, реализующая f.

В дальнейшем будем синтезировать схемы над базисом $\{\&, \vee, \neg\}$ и считать, что у этих ФЭ веса сложности 1, 1 и 0 соответственно. В рамках низкоресурсной криптографии интересно минимизировать сложность схемы, поэтому приведем несколько связанных утверждений о некоторых методах синтеза из [4].

Утв.1 *При синтезе на основе СДНФ* $L(n) \leq n \cdot 2^n$

Утв.2 *При синтезе с улучшенной реализацией конъюнкций* $L(n) \leq 3 \cdot 2^n + n - 5$

Утв.3 *При синтезе путем разложения по переменным* $L(n) \leq 3 \cdot 2^n - 4$

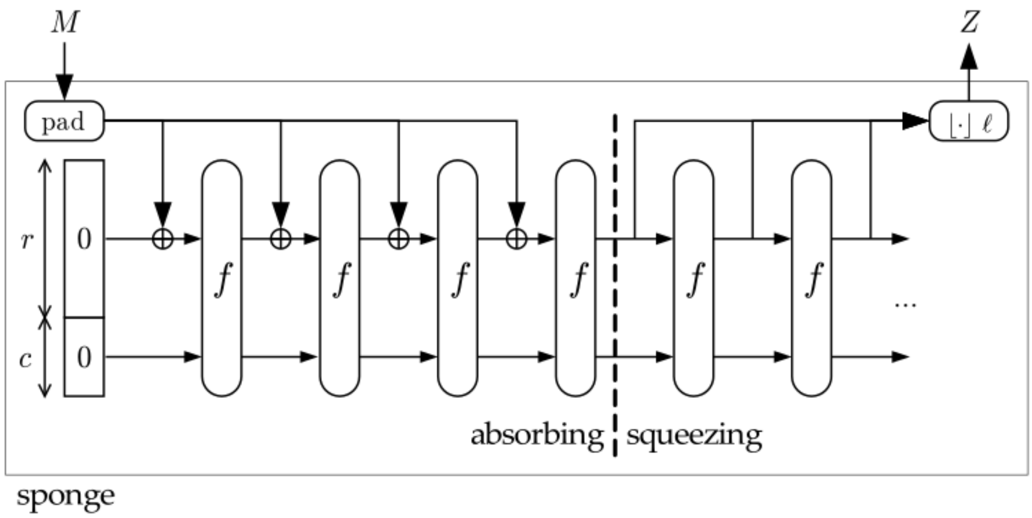
Утв.4 *При синтезе методом Шеннона* $L(n) \lesssim \frac{8 \cdot 2^n}{n}$

Утв.5 *При синтезе методом Лупанова* $L(n) \lesssim \frac{2^n}{n}$

3.5 Криптографические структуры

Регистровый сдвиг с линейной обратной связью (РСЛОС, LFSR) [6] – это регистровый сдвиг, входной бит которого линейно зависит от битов предыдущего состояния.
Применяется для генерации псевдослучайных последовательностей битов. При аппаратной реализации может задаваться полиномами.

Блочный шифр [7] – это шифр, оперирующий блоками (дополненного по необходимости) текста равной и фиксированной длины. Один из видов блочных шифров – *SP-сеть* (substitution-permutation network, SPN) – чередование нелинейной подстановки и линейной перестановки на каждом раунде.
S-блок [8] – это нелинейное преобразование, принимающее на вход n бит и возвращающее m бит. Данное преобразование проще всего представлять как таблицу подстановок размером n × m или как булев оператор $(f_1(x_1, ..., x_n), ..., f_m(x_1, ..., x_n)) : E_2^n \to E_2^m$.
Линейная перестановка (bit shuffling) – этап работы криптографического алгоритма, на котором входные биты перемешиваются по какому-то правилу в несколько этапов. Этот слой нужен для распространения влияния изменения одного бита входного текста на выходные биты и применяется между нелинейными этапами в течение нескольких *раундов*.
Губковый алгоритм (Spodge-based algorithm) [9] – это семейство блочных шифров, представимых в виде автоматов с начальным состоянием конечной длины, работа которого схематически может быть изображена следующим образом:



Вначале исходное сообщение M дополняется до нужной длины (*padding*) и разбивается на блоки M_i длины r. Начальное состояние S имеет длину $r + c = b$.
Первая часть работы алгоритма, *вбирание* (*absorbing*), представляет собой последовательное побитовое сложение S с M_i и применение повторяющегося от раунда к раунду функционального блока алгоритма.
Далее алгоритм переходит в режим *выпускания* (*squeezing*) кусочков шифр-текста, которые могут быть подвержены какой-то дополнительной постобработке.

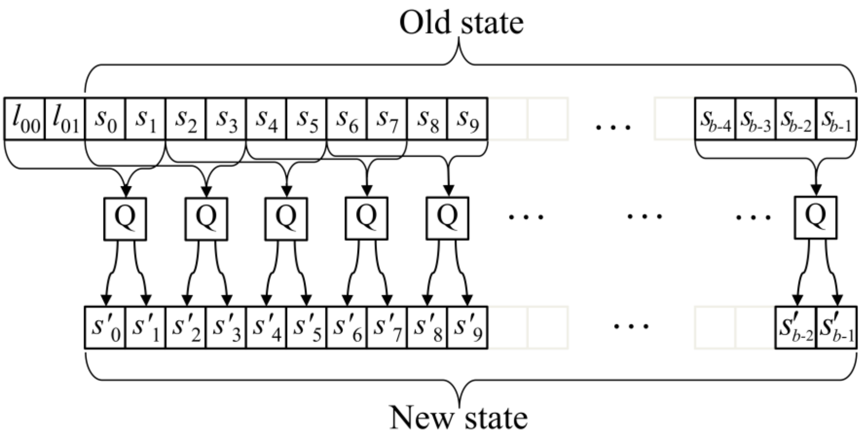
AEAD-режимы блочного шифрования (Authenticated Encryption with Associated Data, «аутентифицированное шифрование с присоединёнными данными») [7] – класс блочных режимов шифрования, при котором часть сообщения шифруется, часть остается открытой, и всё сообщение целиком аутентифицировано.

3.6 Описание алгоритма GAGE-InGAGE [1]

GAGE – это семейство хеш-функций на основе губки с состояниями S от 232 до 576 бит и внедрением блоков сообщений размером r = 8, 16, 32 и 64 бита. Функциональный блок имеет структуру SP-сети с одним очень легким 4-2 битным S-блоком и дешевой аппаратной проводкой.

Нелинейная подстановка

На этом этапе используется 4-2 битный s-блок Q, который применяется параллельно к b-битному состоянию «со сдвигом» по 2 бита. Так, нелинейная подстановка представляет из себя большой (b+2)-b битный s-блок, реализующий квазигрупповое d-преобразование. На каждом раунде алгоритма используется свой 2-битный лидер l_i.



Представления Q:

- Подстановка:

Input:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Output:	1	0	3	2	0	2	1	3	2	3	0	1	3	1	2	0

- Таблица умножения квазигруппы ($a||b := x_1x_2||x_3x_4$):

*	0	1	2	3
0	1	0	3	2
1	0	2	1	3
2	2	3	0	1
3	3	1	2	0

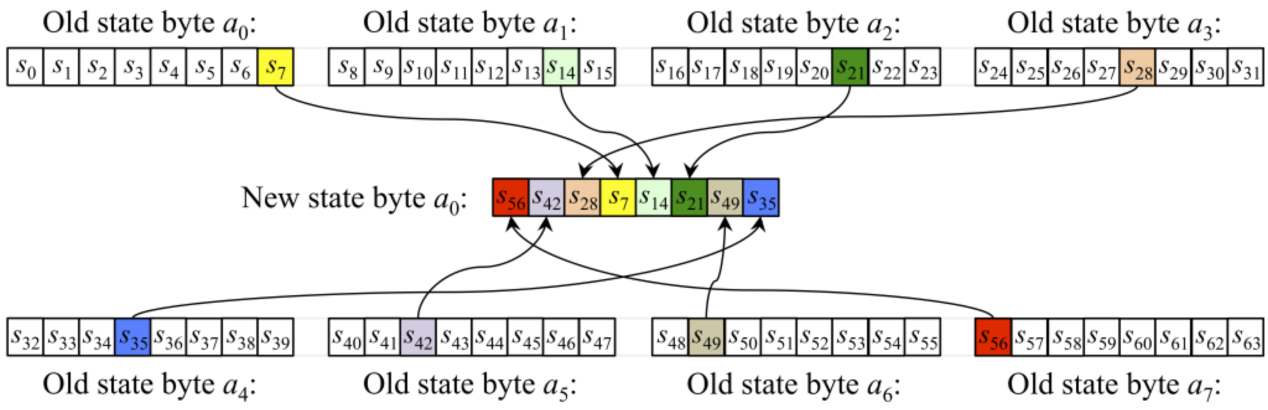
- Функциональный вид $Q(x_1, x_2, x_3, x_4) = (f_1, f_2)$ в $(P_2, +, *)$:
  $Q(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_2x_3 + x_2x_4, 1 + x_1 + x_2 + x_2x_3 + x_4 + x_2x_4)$

Линейный уровень

На этом этапе b-битное состояние $S = (s_0, ...s_{b-1})$ представляется в виде последовательности байтов $A = (a_0, ..., a_{b/8-1})$.

Тогда $\tilde{a}_i := \pi_8([a_{(i+j) \mod b/8}[7-j], j = 0, ..., 7]), i = 0, ..., b/8-1$,

где $\pi_8 := \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 5 & 8 & 7 & 6 & 2 & 4 \end{pmatrix}$. Один из слоев этой перестановки показан ниже:



Релизация в губковом формате

Так, функциональный блок алгоритма представляет собой:

procedure QPERMUTATION($A = \{a_0, \ldots, a_{b/2-1}\}$, `ROUNDS`)
  D-TRANSFORMATION(l_0, A)
  for $i = 1$ to `ROUNDS - 1` **do**
    BIT-SHUFFLING(A)
    D-TRANSFORMATION(l_i, A)
  endfor

Тогда для состояния автомата $S = S_r||S_c$, $b = r + c$, дополненного сообщения $M_{pad} = M_1||...||M_\mu$, разбитого на μ блоков длины r:

- «Вбирание»:

  for $i = 1$ to μ **do**
    $S \leftarrow (S_r$ `XOR` $M_i) \; || \; S_c$
    $S \leftarrow$ QPERMUTATION(S, `ROUNDS`)
  endfor

- «Выжимание»:

  $H \leftarrow \varepsilon$
  for $i = 1$ to $\frac{|Hash|}{r}$ **do**
    $H \leftarrow H \; || \; S[0, \ldots, r-1]$
    $S \leftarrow$ QPERMUTATION(S, `ROUNDS`)
  endfor
  $Hash \leftarrow H$

Раунды

Авторы рекомендуют использовать фиксированный массив лидеров $l_0, ...l_{ROUNDS-1}$ длины 32:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
l_i	0	3	0	3	0	1	2	3	2	1	2	3	2	1	2	3
i	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
l_i	0	1	2	3	2	3	0	3	2	3	2	3	2	1	2	1

С помощью сдвига, определенного $x^8+x^6+x^5+x^4+1$, сгенерирована непериодическая битовая последовательность длины 255, по которой строится 2-битовая последовательность так: чередуются множества $\{0, 1\}$, $\{2, 3\}$ и в зависимости от бита в них выбирается элемент.

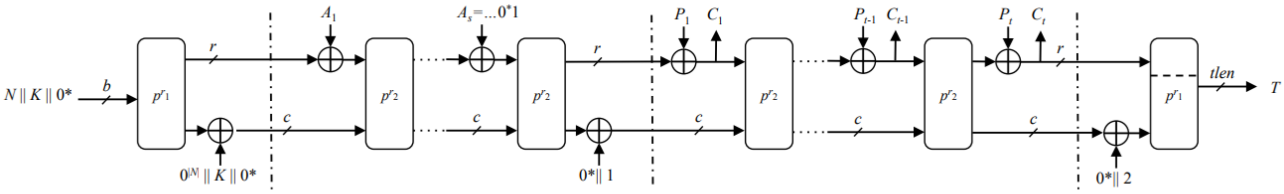
Спецификация InGAGE

InGAGE использует стандартный AEAD-режим от 4-х параметров, который описывается функциями шифрования и расшифровывания

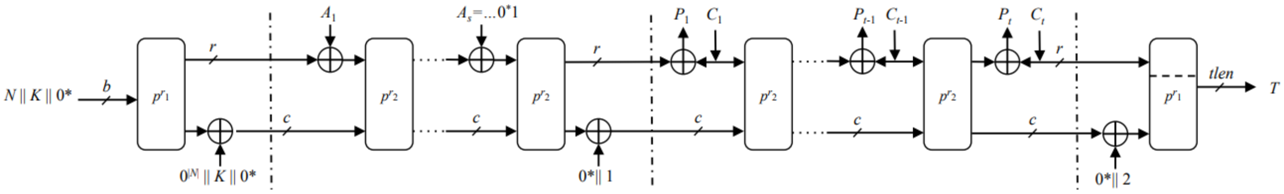
$$\mathcal{E}(K, N, A, P) = (C, T) \qquad \mathcal{D}(K, N, A, C, T) \in \{P, Invalid\},$$

где K – ключ, N – одноразовый номер, A – связанные данные,
P – открытый текст, C – шифртекст той же длины, что открытый,
T – аутентификационный тег, $r_1 = ROUNDS$, $\quad r_2 = ROUNDS/2$.

Процесс шифрования:



Процесс дешифрования:



4 Результаты: методы построения, построенные схемы, характеристики

Основной интерес представлял синтез схем, реализующих лидеры и квазигрупповой s-блок. Для минимизации сложности схемы разумно [4] представлять реализуемые функции в виде minДНФ или хотя бы сокращенной ДНФ и стараться максимально объединять общие части выражений, уменьшая число функциональных элементов.

Для облегчения построения минимальной ДНФ функций и самопроверки была написана программа, приводящая список простых импликант, входящих в сокДНФ, по методу Квайна-Мак-Класки. Ознакомиться с ней, как и с другими созданными в процессе программами, можно на моем github [10]. Опишем вывод этой программы:
Вначале наборы значений, разбиты на группы по числу переменных без отрицания. Далее показан результат работы алгоритма: максимальные грани функции (в примере ниже они соответствуют конъюнкциям x_2x_3, x_1x_3 и x_1x_2 соответственно). Отбрасывание не ядровых (для получения ДНФ Квайна) и возможно других импликант (для получения minДНФ) проводилось вручную (в примере ниже все импликанты ядровые, поэтому входят в minДНФ $Maj(x_1, x_2, x_3)$).

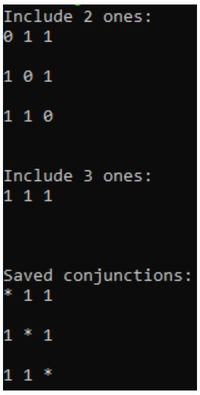


Рис.1[11]: *Пример использования программы для* $Maj(x_1, x_2, x_3)$.

Синтез константных лидеров, зависящих от раунда, представляется как минимум двумя способами: с помощью 5-битного инкремента, передающего значение в функцию, вычисляющую значения лидеров, или в виде автомата, реализующего 32 (по количеству раундов) задержки для возвращения соответствующей лидерной константы. Положив вес сложности каждой задержки равным 3-7 логических ячеек, приходим к выводу, что для оптимизации сложности лучше реализовывать инкремент и функцию.

Описать инкремент в качестве конечного автомата можно следующим образом:

$$\begin{cases} \vec{q}(1) = (0, 0, 0, 0, 0) \\ \vec{q}(t+1) = \vec{q}(t) + 1(mod 32) \\ \vec{y}(t) = \vec{q}(t) \end{cases}$$

Нужно реализовать элемент задержки G_i и прибавление 1 к числу $t = t_4||t_3||t_2||t_2||t_0$, обозначающему номер раунда. Так, инкремент реализован со сложностью 15 логических единиц и 5 задержек. При вычислении глубины задержки игнорируются, их входы считаются выходами схемы, выходы – входами схемы. Так, глубина инкремента равна 5.

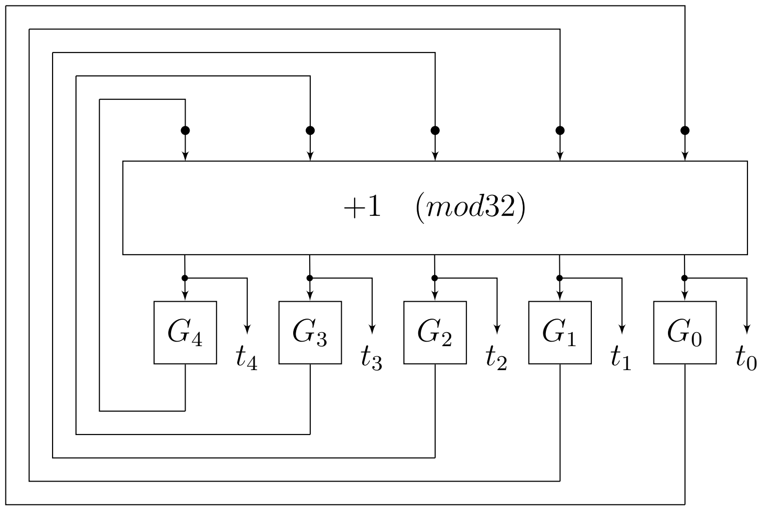


Рис.2: *Общий вид инкремента, отсчитывающего номер раунда.*

Функцию, реализующую двубитные лидерные константы, можно описать так:
$l_1(t_4,t_3,t_2,t_1,t_0) = \bar{t}_3(t_4t_2\bar{t}_1 \vee t_0) \vee \bar{t}_4 \vee \bar{t}_2t_0) \vee t_3(t_4\bar{t}_2 \vee \bar{t}_0) \vee t_1(\bar{t}_4t_2 \vee \bar{t}_2t_0 \vee t_4\bar{t}_2);$
$l_0(t_4,t_3,t_2,t_1,t_0) = t_0.$
Глубина такой реализации составляет 6, сложность – 16 логических единиц.

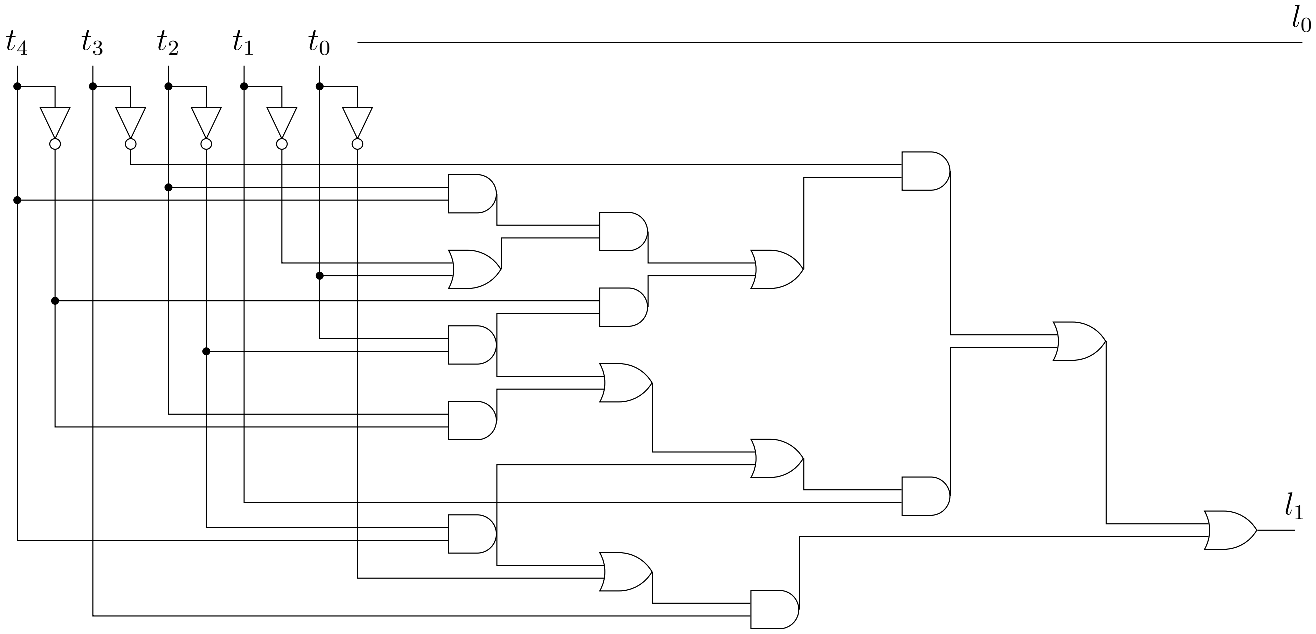


Рис.3: *Схема, реализующая генератор 32 2-битных лидеров.*

Описание этой схемы на Verilog:

```
module leader_generator (t, l);
    input [4:0] t;
    output [1:0] l;
    wire common1 = ~t[2] & t[0];
    wire common2 = t[4] & ~t[2];
    assign l[1] = ~t[3] & (t[4] & t[2] & (~t[1] | t[0]) | ~t[4] & common1)
        | t[3] & (common2 | ~t[0]) | t[1] & (~t[4] & t[2] | common1 | common2);
    assign l[0] = t[0];
endmodule
```

Напомним, что в функциональном виде выходные биты 4-2 битного s-блока Q представляются следующим образом:
$Q(x_3, x_2, x_1, x_0) = (x_3 + x_1 + x_2x_1 + x_2x_0, 1 + x_3 + x_2 + x_2x_1 + x_0 + x_2x_0) =: (f_1, f_2)$.

Запишем minДНФ этих функций (состоящие только из ядровых импликант) и вынесем за скобки общие части функций:
$f_1(x_3, x_2, x_1, x_0) = \bar{x}_3\bar{x}_2x_1 \vee \bar{x}_3x_2x_0 \vee x_3\bar{x}_2\bar{x}_1 \vee x_3x_2\bar{x}_0 = \bar{x}_2(\bar{x}_3x_1 \vee x_3\bar{x}_1) \vee x_2(\bar{x}_3x_0 \vee x_3\bar{x}_0)$
$f_2(x_3, x_2, x_1, x_0) = \bar{x}_3\bar{x}_2\bar{x}_0 \vee \bar{x}_3x_2x_1 \vee x_3\bar{x}_2x_0 \vee x_3x_2\bar{x}_1 = x_2(\bar{x}_3x_1 \vee x_3\bar{x}_1) \vee \bar{x}_2(\bar{x}_3\bar{x}_0 \vee x_3x_0)$.

Сложность схемы, реализующей s-блок Q, оказывается равна 15, глубина - 5. Тогда весь уровень нелинейной подстановки имеет глубину 5 (в силу возможности параллельного вычисления квазигруппового d-преобразования) и сложность $\frac{b}{2} \cdot 15 = 1920$ при $b = 256$ (рекомендуемое авторами алгоритма значение длины состояния S).

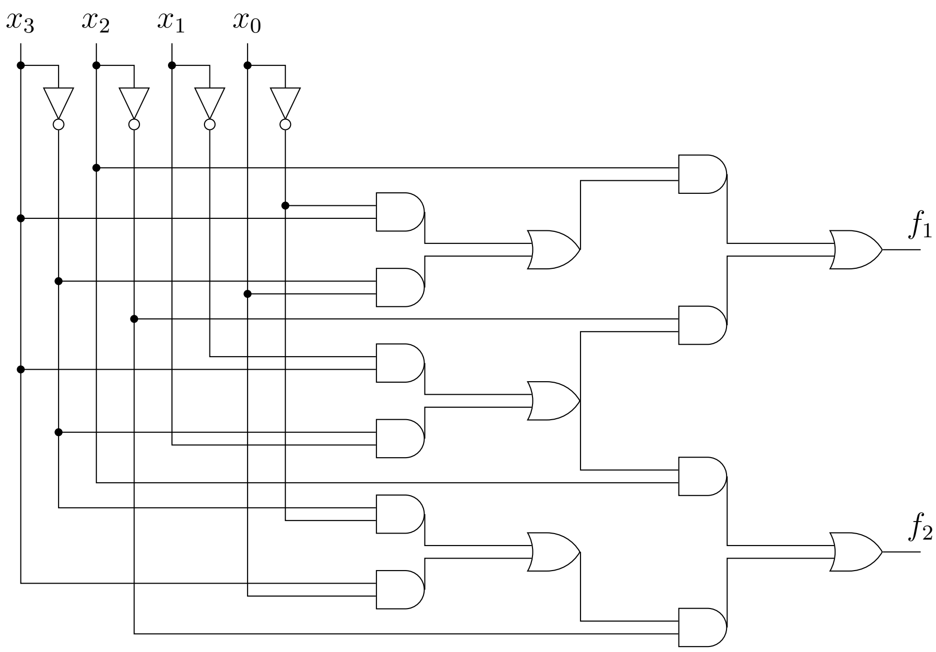


Рис.4: *Схема, реализующая 4-2 битный s-блок Q.*

Описание этой схемы на Verilog:

```
module Q(x, Qx);
    input [3:0] x;
    output [1:0] Qx;
    wire common_block = ~x[3] & x[1] | x[3] & ~x[1];
    assign Qx[1] = ~x[2] & common_block | x[2] & (~x[3] & x[0] | (x[3] & ~x[0]);
    assign Qx[0] = x[2] & common_block | ~x[2] & (~x[3] & ~x[0] | x[3] & x[0]);
endmodule
```

Так, синтезированая схема, состоящая из последовательно соединенных инкремента, генератора лидерных констант и слоя параллельно подсоединенных к входному блоку и выходу генератора лидерных констант Q-блоков (см. нелинейная подстановка GAGE), реализует один раунд алгоритма без учета линейного уровня. С учетом того, что этот уровень не несет логической нагрузки и только переставляет биты непосредственно проводами, общая сложность полученной схемы, реализующей один раунд, составляет $15 + 16 + \frac{b}{2} \cdot 15 = 1951$ логических единиц и 5 задержек и имеет глубину $5 + 6 + 5 = 16$.

5 Заключение

В рамках этой курсовой работы я освоила некоторые идеи и понятия квазигрупповой алгебры и криптографии, изучила документацию GAGE-InGAGE и в итоге разобралась в ней на уровне достаточном, чтобы с помощью материалов специального курса кафедры МАТИС и научно-исследовательского семинара А.В.Галатенко предложить вполне низкоресурсные варианты аппаратной реализации элементов этого алгоритма.

Представляют интерес следующие направления дальнейшего развития:

- описание на Verilog и тестирование всей шифрующей схемы;
- использование мультиплексоров для сокращения числа s-блоков Q в рамках одного раунда (последовательное применение каждого блока 2+ раза в раунд) для возможного улучшения сложности;
- реализация 2+ раундов за такт;
- сравнение GAGE-InGAGE с похожими алгоритмами этой направленности.

Также в ходе работы возник запрос на написание вспомогательной программы, подсчитывающей глубину и сложность схемы, описанной на Verilog, и поддерживающей использование вложенных функциональных блоков, и интерес к обзору существующих оптимизаторов глубины схемы и программ, синтезирующих код на Verilog по какому-то альтернативному описанию алгоритма.

Список литературы

[1] D. Gligoroski, H. Mihajloska, D. Otte, M. El-Hadedy, "GAGE and InGAGE", **v1.03** (2019), http://gageingage.org/.

[2] https://csrc.nist.gov/projects/lightweight-cryptography.

[3] В. Д. Белоусов, "Основы теории квазигрупп и луп", *Наука*, 1967.

[4] С. В. Яблонский, "Введение в дискретную математику", *Высшая школа*, 2001.

[5] В. Б. Кудрявцев, С. В. Алешин, А. С. Подколзин, "Введение в теорию автоматов", *Наука*, 1985.

[6] U. Jetzek, "Galois Fields, Linear Feedback Shift Registers and their Applications", 1996.

[7] A. Menezes, P. van Oorschot, S. Vanstone, "Handbook of Applied Cryptography", *CRC Press*, 1996.

[8] Е. А. Курганов, "Об аппаратной реализации сбалансированных S-блоков", *Программная инженерия ISSN 2220-3397*, **1** (2021), 8–20.

[9] G. Bertoni, J. Daemen, M. Peeters, G. van Assche, "Keccak sponge function family main document", 2009.

[10] https://github.com/MatsuMizu.

[11] https://imgur.com/a/xKTksbc.