

Гибридная и полностью битово-параллельная: две стратегии реализации MGM

Анатолий Соловьев, Денис Фомин

CTCrypt 2026, 1-5 июня, г. Минск, Республика Беларусь

Описание MGM

MGM (Multilinear Galois Mode) - режим работы блочного шифра.

Режим MGM определяется в стандарте

Р 1323565.1.026-2019 Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров, реализующие аутентифицированное шифрование.

RFC 9058 в 2021 году.

MGM определяется с блочными шифрами «Магма» с длиной блока $n = 64$ бит или «Кузнечик» $n = 128$ бит.

Блочные шифры определяются в стандарте

ГОСТ 34.12-2015 Информационная технология. Криптографическая защита информации. Блочные шифры.

MGM относится к семейству AEAD-режимов и обеспечивает конфиденциальность и имитозащиту сообщений. При этом сообщения могут состоять из одной или двух частей. Если сообщение состоит из двух частей, то первая часть содержит дополнительные имитозащищаемые данные обозначается A , а вторая часть называется открытым текстом и обозначается P .

Параметром режима MGM является длина имитовставки S , где $32 \leq S \leq n$.

- V_s – множество битовых строк длины s ;
- $K \in V_k$ – ключ шифрования;
- $nonce \in V_{n-1}$ – уникальный вектор для каждого сообщения при фиксированном ключе;
- $E_K : V_n \rightarrow V_n$ – отображение, реализующее функцию зашифрования с использованием ключа $K \in V_k$;
- $\|$ – конкатенация битовых строк; $a = (a_1, \dots, a_{s1}) \in V_{s1}$, $b = (b_1, \dots, b_{s2}) \in V_{s2}$, то $a\|b = (a_1, \dots, a_{s1}, b_1, \dots, b_{s2}) \in V_{s1+s2}$;
- $Int_s : V_s \rightarrow \mathbb{Z}_{2^s}$ – отображение, ставящее в соответствие битовой строке $a = (a_1, \dots, a_s) \in V_s$ число $Int_s(a) = 2^{s-1}a_1 + \dots + 2^0a_s$;
- $Vec_s : \mathbb{Z}_{2^s} \rightarrow V_s$ – отображение, обратное Int_s ;
- $\boxplus_{n/2}$ – сложение в кольце $\mathbb{Z}_{2^{n/2}}$;
- $incr_l : V_n \rightarrow V_n$ – отображение, сопоставляющее строке $L\|R$, где $L, R \in V_{n/2}$, строку $incr_l(L\|R) = Vec_{n/2}(Int_{n/2}(L) \boxplus_{n/2} 1)\|R$;
- $incr_r : V_n \rightarrow V_n$ – отображение, сопоставляющее строке $L\|R$, где $L, R \in V_{n/2}$, строку $incr_r(L\|R) = L\|Vec_{n/2}(Int_{n/2}(R) \boxplus_{n/2} 1)$;

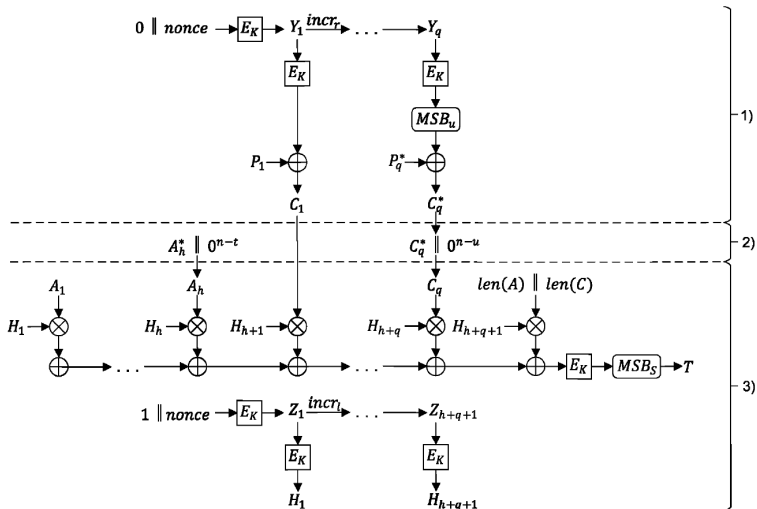
- $\otimes : V_n \times V_n \rightarrow V_n$ – отображение, ставящее двум строкам $a = (a_1, \dots, a_n) \in V_n$, $b = (b_1, \dots, b_n) \in V_n$ строку $c = a \otimes b = (c_1, \dots, c_n)$, $c \in V_n$; строка c соответствует многочлену $c(x) = c_1 x^{n-1} + \dots + c_{n-1} x + c_n$, который является результатом умножения двух многочленов $a(x) = a_1 x^{n-1} + \dots + a_{n-1} x + a_n$ и $b(x) = b_1 x^{n-1} + \dots + b_{n-1} x + b_n$ в поле $GF(2^n)$; для $n = 64$ задан порождающий многочлен $f(x) = x^{64} + x^4 + x^3 + x + 1$, для $n = 128$ задан порождающий многочлен $f(x) = x^{128} + x^7 + x^2 + x + 1$.

Дополнительные обозначения (для наглядности описания реализаций MGM)

- w – длина машинного слова (битового среза), $w = 256$, т.к. AVX2.
- $(\dots)^T$ – транспонирование данных, из контекста понятно в классическое или bitslice представление.
- $E_K^{bs} : (V_{256})^n \rightarrow (V_{256})^n$ – зашифрование на ключе K в bitslice-режиме;
- $\oplus^{bs} : (V_{256})^n \times (V_{256})^n \rightarrow (V_{256})^n$ – побитовое сложение (операция XOR) в bitslice-режиме;
- $\otimes^{bs} : (V_{256})^n \times (V_{256})^n \rightarrow (V_{256})^n$ – умножение в поле $GF(2^n)$ по схеме Карацубы в bitslice-режиме;
- $\boxplus_{n/2}^{bs} : (V_{256})^{n/2} \times (V_{256})^{n/2} \rightarrow (V_{256})^{n/2}$ – сложение в кольце $\mathbb{Z}_{2^{n/2}}$ в bitslice режиме;
- $incr_l^{bs}, incr_r^{bs}$ – добавление 1 к соответствующей половине векторов в bitslice-режиме.
- $incr256_l^{bs}, incr256_r^{bs}$ – добавление 256 к соответствующей половине векторов в bitslice-режиме.

Верхний индекс sc в обозначении операции подчеркивает, что операция выполняется скалярно. Например, E_K^{sc} в точности операция E_K .

Зашифрование сообщения в режиме MGM



Битово-параллельная реализация

Bitslice – способ программной реализации, который предполагает разбиение алгоритма на логические побитовые операции (**AND**, **OR**, **XOR**, **NOT**). При этом, использование N-битных машинных слов позволяет выполнять параллельную работу с набором из N данных, что повышает производительность программы. Машинные слова можно называть битовыми срезами.

Машинное слово – фрагмент данных фиксированного размера, обрабатываемый как единое целое с помощью набора команд.

На эффективность bitslice влияет разрядность используемых машинных слов. Чем больше разрядность, тем большее количество входных данных можно обработать при сохранении количества инструкций.

Реализация в рамках данной работы использует векторные инструкции AVX2, соответственно, машинными словами являются 256-разрядные векторные регистры.

Например, 256 чисел, каждое из которых занимает 8 бит можно хранить `uint8_t a[256];` // `a[0]` - 0-е число, `a[1]` - 1-е число, ..., - классический способ В bitslice эти же 256 чисел можно хранить

`__m256i b[8];` // `b[0]` хранит все 0-е биты чисел, ..., `b[7]` все 7-е биты
Элементы `b[0]`, `b[1]`, ..., `b[7]` являются битовыми срезами.

Гибридная реализация MGM

Гибридная реализация режима MGM применяется к одному входному сообщению.

Вход: A , P , $nonce$, ключ K , длины l_A , l_P .

Выход: шифртекст C (перезаписывает P) и имитовставка im .

В данной версии для эффективности некоторые вычисления реализованы без применения bitslice, поэтому и называется гибридная.

Вычисления без применения bitslice:

$$Y_1 = E_K(0 || nonce)$$

$$Z_1 = E_K(1 || nonce)$$

$$H_{h+q+1} \otimes (\text{len}(A) || \text{len}(C)),$$

а также финальное зашифрование (только зашифрование $E_K(\dots)$)

$$E_K\left(\bigoplus_{j=1}^h H_j \otimes A_j \oplus \bigoplus_{i=1}^q H_{h+i} \otimes C_i \oplus H_{h+q+1} \otimes (\text{len}(A) || \text{len}(C))\right)$$

Гибридная реализация MGM

Зашифрование сообщения (A, P) состоит из следующих этапов. Обработка части $A = A_1, \dots, A_h$. Последний блок имеет длину $t \leq n$.

Инициализация: $curImit^{(i)} = \vec{0}, i = 1, \dots, n$.

- 1 Подготовка счётчика: $Z_1 = E_K^{sc}(1 || nonce)$, $Z_1 = (Z_{1L} || Z_{1R})$. Далее в bitslice режиме формируются $Z_2 = incr_l(Z_1), \dots, Z_{256} = incr_l(Z_{255})$:
 $(Z_1, \dots, Z_{256})^T \leftarrow ((Z_{1L}, \dots, Z_{1L})^T \boxplus^{bs} (0, 1, \dots, 255)^T) || (Z_{1R}, \dots, Z_{1R})^T$.
Счётчик массив из n 256-битных векторов $Z1_1, \dots, Z1_n \leftarrow (Z_1, \dots, Z_{255})^T$.
- 2 В цикле ($s = 1, s \leq h, s = s + 256$), обрабатываются по 256 блоков $A_s, \dots, A_{s+255}$, выполняются следующие шаги.
 - 1 Копирование: $Z2_1, \dots, Z2_n \leftarrow Z1_1, \dots, Z1_n$
 - 2 Добавление 256: $Z1_1, \dots, Z1_n \leftarrow incr_{256}^{bs}(Z1_1, \dots, Z1_n)$
 - 3 Формирование ключей: $(H_s, \dots, H_{s+255})^T \leftarrow E_K^{bs}(Z2_1, \dots, Z2_n)$
 - 4 Умножение по схеме Карацубы и добавление к текущему массиву $curImit^{(1)}, \dots, curImit^{(n)} \leftarrow (curImit^{(1)}, \dots, curImit^{(n)}) \oplus^{bs} ((H_s, \dots, H_{s+255})^T \otimes^{bs} (A_s, \dots, A_{s+255})^T)$.
На последней итерации цикла - блок A_h дополняется $n - t$ нулевыми битами (если длина A_h меньше n).
Реализация допускает зацепление $(h - s + 1)$ последних блоков $A_s, \dots, A_h$ и $256 - (h - s + 1)$ первых блоков шифртекста $C_1, \dots, C_{256-(h-s+1)}$, см. параметр $AwithP$.

Гибридная реализация MGM

Обработка $P = P_1, \dots, P_q$, первые $q - 1$ блоков длины n , последний длины $u \leq n$.

- 3 Подготовка счётчика: $Y_1 = E_K^{sc}(0 || \text{nonce})$, $Y_2 = \text{incr}_r(Y_1)$, $\dots$, $Y_{256} = \text{incr}_r(Y_{255})$. Счётчиком является массив из n 256-разрядных векторов: $Y1_1, \dots, Y1_n = (Y_1, \dots, Y_{256})^T$.
- 4 В цикле ($s = 1, s \leq q, s = s + 256$), обрабатываются по 256 блоков $P_s, \dots, P_{s+255}$, выполняются следующие шаги.
 - 1 Копирование счётчика: $Y2_1, \dots, Y2_n \leftarrow Y1_1, \dots, Y1_n$.
 - 2 Добавление 256: $Y1_1, \dots, Y1_n \leftarrow \text{incr}_{256}^{bs}(Y1_1, \dots, Y1_n)$.
 - 3 Зашифрование счётчика и наложение гаммы на открытый текст:
 $C_s, \dots, C_{s+255} \leftarrow ((P_s, \dots, P_{s+255})^T \oplus^{bs} E_K^{bs}(Y2_1, \dots, Y2_n))^T$
 - 4 Умножение по схеме Карацубы и добавление к текущему массиву $\text{curImit}^{(1)}, \dots, \text{curImit}^{(n)} \leftarrow (\text{curImit}^{(1)}, \dots, \text{curImit}^{(n)}) \oplus^{bs} (H_{h+s}, \dots, H_{h+s+255})^T \otimes^{bs} ((P_s, \dots, P_{s+255})^T \oplus^{bs} E_K^{bs}(Y2_1, \dots, Y2_n))$, формирование $(H_s, \dots, H_{s+255})^T$ описано ранее, в обработке части A. В последнем блоке C_q перед умножением необходимо обнулить $n - u$ бит.
- 5 Имитовставка $im = E_K^{sc}(\bigoplus_{i=1}^{256} v_i \oplus^{sc} H_{h+q+1} \otimes^{sc} (\text{len}(A) || \text{len}(C)))$,
 $v_1, \dots, v_{256} = (\text{curImit}^{(1)}, \dots, \text{curImit}^{(n)})^T$.

Полностью битово-параллельная реализация MGM

Полностью битово-параллельная реализация параллельно обрабатывает $1 \leq M \leq 256$ сообщений (при использовании векторных инструкций AVX2). Предложенная реализация работает с сообщениями $(A^{(i)}, P^{(i)})$ одинаковой длины $l_A = \text{len}(A^{(i)})$, $l_P = \text{len}(P^{(i)})$, $i = 1, \dots, M$.

Вход: наборы $(A^{(i)}, P^{(i)}, \text{nonce}^{(i)})$, $i = 1, \dots, M$, ключ K , длины l_A , l_P .

Выход: шифртекст $C^{(i)}$ и имитовставка $\text{im}^{(i)}$, $i = 1, \dots, M$.

В работе реализуется стратегия: один ключ K - несколько сообщений $(A^{(i)}, P^{(i)}, \text{nonce}^{(i)})$, $i = 1, \dots, M$.

Несложно реализовать стратегию: один ключ $K^{(i)}$ - одно сообщение $(A^{(i)}, P^{(i)}, \text{nonce}^{(i)})$, $i = 1, \dots, M$.

Сложнее реализовать стратегию: сообщения разной длины, т.е. когда существует $l_{A^{(i)}} \neq l_{A^{(j)}}$ или $l_{P^{(i)}} \neq l_{P^{(j)}}$, для $i \neq j$, $i, j = 1, \dots, M$. Например, см. bitslice реализацию функции хэширования «Стрибог» в работе¹, в которой не накладывается ограничение в виде равенства длин сообщений.

¹А.С.Соловьев, Д.Б.Фомин, Битово-параллельная реализация алгоритмов «Стрибог» и «Кузнечик», СТCrypt 2025.

Полностью битово-параллельная реализация MGM

Зашифрование сообщений $(A^{(1)}, P^{(1)}), \dots, (A^{(M)}, P^{(M)})$ состоит из следующих этапов. Обработка $A^{(1)}, \dots, A^{(M)}$, т.е. 1-й части сообщений, где $A^{(i)} = A_1^{(i)}, \dots, A_h^{(i)}$ состоит из $h - 1$ блоков длины n , длина последнего блока $t \leq n, i = 1, \dots, M$.

Инициализация: $curImit^{(i)} = \vec{0}, i = 1, \dots, n$.

❶ Подготовка счётчика:

$Z_1^{(1)}, \dots, Z_1^{(n)} \leftarrow E_K^{bs}((1||nonce^{(1)}, 1||nonce^{(2)}, \dots, 1||nonce^{(M)})^T)$ - массив из n 256-разрядных векторов (т.к. AVX2), $Z_1^{(i)}$ содержит все i -е биты результата зашифрования, $i = 1, \dots, n, n \in \{64, 128\}$ - длина блока шифра.

❷ В цикле $s = 1, \dots, h$, где h - количество блоков, выполняются следующие шаги.

❶ Копирование: $Z2_s^{(1)}, \dots, Z2_s^{(n)} \leftarrow Z_s^{(1)}, \dots, Z_s^{(n)}$.

❷ Добавление единицы: $Z_{s+1}^{(1)}, \dots, Z_{s+1}^{(n)} \leftarrow incr_l^{bs}(Z_s^{(1)}, \dots, Z_s^{(n)})$.

❸ Формирование ключей: $H_s^{(1)}, \dots, H_s^{(n)} \leftarrow E_K^{bs}(Z2_s^{(1)}, \dots, Z2_s^{(n)})$.

❹ Умножение по схеме Карацубы и добавление к текущему массиву:
 $curImit^{(1)}, \dots, curImit^{(n)} \leftarrow (curImit^{(1)}, \dots, curImit^{(n)}) \oplus^{bs} (H_s^{(1)}, \dots, H_s^{(n)}) \otimes^{bs} (A_s^{(1)}, \dots, A_s^{(M)})^T$. На последней итерации ($s = h$) блоки $A_h^{(1)}, \dots, A_h^{(M)}$ дополняются $(n - t)$ нулями перед транспонированием.

Полностью битово-параллельная реализация MGM

Обработка $P^{(1)}, \dots, P^{(M)}$, $P^{(i)} = P_1^{(i)}, \dots, P_q^{(i)}$ состоит из q блоков, первые $q - 1$ блоков длины n , длина последнего блока $u \leq n$, $i = 1, \dots, M$.

- 3 Подготовка счётчика $(Y_1^{(1)}, \dots, Y_1^{(n)}) = E_K^{bs}((0||nonce^{(1)}, \dots, 0||nonce^{(M)})^T)$ - массив из n векторов длины 256 бит (AVX2), где $Y_1^{(i)}$ содержит все i -е биты результата зашифрования, $i = 1, \dots, n$, $n \in \{64, 128\}$ - длина блока шифра.
- 4 В цикле $s = 1, \dots, q$ выполняются следующие шаги.
 - 1 Копирование счётчика $(Y2_s^{(1)}, \dots, Y2_s^{(n)}) \leftarrow (Y_s^{(1)}, \dots, Y_s^{(n)})$.
 - 2 Добавление единицы $(Y_{s+1}^{(1)}, \dots, Y_{s+1}^{(n)}) \leftarrow incr_r^{bs}(Y_s^{(1)}, \dots, Y_s^{(n)})$.
 - 3 Зашифрование счётчика и наложение гаммы на открытый текст
 $C_1, \dots, C_n \leftarrow (P_s^{(1)}, \dots, P_s^{(M)})^T \oplus^{bs} E_K^{bs}(Y2_s^{(1)}, \dots, Y2_s^{(n)})$,
 $C_s^{(1)}, \dots, C_s^{(M)} \leftarrow (C_1, \dots, C_n)^T$ - сохранение зашифрованных блоков.
 - 4 Умножение по схеме Карацубы $curImit^{(1)}, \dots, curImit^{(n)} \leftarrow curImit^{(1)}, \dots, curImit^{(n)} \oplus^{bs} (H_{h+s}^{(1)}, \dots, H_{h+s}^{(n)}) \otimes^{bs} (C_1, \dots, C_n)$,
формирование $(H_s^{(1)}, \dots, H_s^{(n)})$ описано ранее, в обработке части A. На последней итерации ($s = q$), если $u < n$, перед умножением необходимо добавить $n - u$ нулевых векторов к $(C_h^{(1)}, \dots, C_h^{(M)})^T$.
- 5 Имитовставки $im^{(1)}, \dots, im^{(M)} = (E_K^{bs}((curImit^{(1)}, \dots, curImit^{(n)}) \oplus^{bs} (H_{h+q+1}^{(1)}, \dots, H_{h+q+1}^{(n)}) \otimes^{bs} (\text{len}(A)||\text{len}(C))))^T$

Транспонирование данных

В основе транспонирования данных - алгоритм из книги², реализует транспонирование битовой матрицы размера 8×8 за 21 инструкцию. В настоящей работе данный алгоритм адаптирован для AVX2 инструкций, что позволяет транспонировать четыре матрицы размера 8×8 за 21 инструкцию, т.к. одна матрица 8×8 занимает 64 бита, четыре - 256 бит. Исходные данные разбиваются на подматрицы размера 32×32 , которые транспонируются за 136 инструкций (операция присваивания = не учитывалась).

Набор из 256 блоков «Магма» имеет размер 256×64 , транспонируется за $8 \cdot 2 \cdot 136 = 2176$ инструкций.

Набор из 256 блоков «Кузнечик» имеет размер 256×128 , транспонируется за $8 \cdot 4 \cdot 136 = 4352$ инструкций.

²Henry S. Warren, Hacker's delight, 2013.

PEIGEN³ - платформа для изучения, реализации и генерации S-боксов. Платформа позволяет находить реализации для S-боксов в различных логических базисах (**AND**, **OR**, **XOR**, **NOT**, **ANDNOT**, ...) с учётом критериев минимизации: Bitslice Gate Complexity (BGC) - количество логических инструкций; Multiplicative Complexity (MC) - мультипликативная сложность, количество инструкций AND, инструкции XOR и NOT считаются «бесплатными»; Depth Complexity - глубина схемы; суммарная стоимость инструкций (вентилей), т.е. каждая инструкция имеет свою стоимость, определенную числом.

Работает с n -битными S-блоками, $3 \leq n \leq 8$.

³Zhenzhen Bao, Jian Guo, San Ling, Yukihiro Sasaki, Peigen – a Platform for Evaluation, Implementation, and Generation of S-boxes, 2019, <https://tosc.iacr.org/index.php/ToSC/article/view/7406/6578>, <https://github.com/peigen-sboxes/PEIGEN>.

Применение PEIGEN

Используя PEIGEN построены bitslice реализации восьми подстановок $\pi_0, \dots, \pi_7$ для «Магма» и трёх преобразований, участвующих в реализации подстановки «Кузнечик». w - разрядность машинного слова ($w = 256$ для AVX2).

Algorithm Битово-параллельная реализация π_0

Require: $\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3], \mathbf{x}[i] \in \mathbb{F}_2^w, i = 0, 1, 2, 3$

Ensure: $\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3] = \pi_0(\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3])$

- 1: $t \leftarrow \mathbf{x}[1]$
- 2: $\mathbf{x}[0] \leftarrow \mathbf{x}[0] \oplus \mathbf{x}[2] \oplus (t \vee \mathbf{x}[3])$
- 3: $\mathbf{x}[1] \leftarrow \mathbf{x}[3] \oplus (\mathbf{x}[0] \wedge \mathbf{x}[2])$
- 4: $\mathbf{x}[2] \leftarrow \mathbf{x}[2] \oplus \mathbf{x}[1]$
- 5: $\mathbf{x}[1] \leftarrow \neg \mathbf{x}[1]$
- 6: $t \leftarrow t \oplus ((\mathbf{x}[0] \wedge \mathbf{x}[1]) \vee \mathbf{x}[2])$
- 7: $\mathbf{x}[3] \leftarrow \mathbf{x}[1] \oplus (\mathbf{x}[0] \vee t)$
- 8: $\mathbf{x}[1] \leftarrow \mathbf{x}[0] \oplus \mathbf{x}[2]$
- 9: $\mathbf{x}[0] \leftarrow \mathbf{x}[2] \oplus (\mathbf{x}[3] \wedge t)$
- 10: $\mathbf{x}[2] \leftarrow \mathbf{x}[1] \oplus \mathbf{x}[3]$
- 11: $\mathbf{x}[1] \leftarrow t \oplus \mathbf{x}[1]$

Сложность реализации подстановок в «Магма»

Сложность битово-параллельной реализации (Bitslice Gate Complexity - BGC) подстановок π_i в «Магме» (в логических операциях)

i	BGC (π_i)	AND	XOR	NOT	OR
0	17	3	10	1	3
1	17	3	8	1	5
2	16	3	9	1	3
3	15	1	8	1	5
4	16	3	8	2	3
5	17	2	9	2	4
6	17	4	9	1	3
7	17	5	9	1	2
Итого	132	24	70	10	28

Сложение с раундовым ключом в алгоритме «Магма»

Операция сложения в кольце $\mathbb{Z}_{2^N}$, где N - натуральное число, может быть эффективно реализована с использованием классической идеи, заключается в представлении чисел в виде битовых векторов и выполнении операций над битами «в столбик». Для «Магма» $N = 32$ и сложность равна $32 \cdot 5 = 160$.

Algorithm Сложение в $\mathbb{Z}_{2^N}$ «в столбик»

```
1: Вход:  $A = (A[0], \dots, A[N-1])$  и  $B = (B[0], \dots, B[N-1])$ ,  $A[i], B[i] \in F_2^w$   
   - битовые срезы, представляющие двоичные записи элементов  $\mathbb{Z}_{2^N}$ .  
2: Выход: Результат сложения result.  
3: carry  $\leftarrow 0$   
4: result  $\leftarrow [0] \times N$   
5: for all  $i = 0$  to  $N - 1$  do  
6:    $s_i \leftarrow A[i] \oplus B[i]$   
7:    $c_i \leftarrow A[i] \wedge B[i]$   
8:   result[i]  $\leftarrow s_i \oplus$  carry  
9:   carry  $\leftarrow c_i \vee (s_i \wedge$  carry)  
10: end for  
11: return result
```

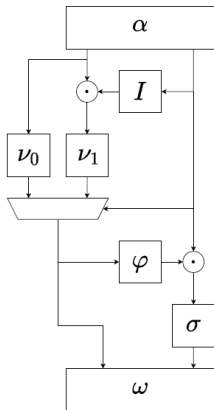
Декомпозиция подстановки π из алгоритма «Кузнечик»

Декомпозиция подстановки π из алгоритма «Кузнечик» представлена в работе⁴.

CTCrypt 2025 $BGC(\pi) = 169$, $Depth(\pi) = 55$.

CTCrypt 2026 $BGC(\pi) = 163$, $Depth(\pi) = 59$.

С использованием PEIGEN построены преобразования I, ν_0, σ с меньшей сложностью. В таблице сравнение BGC (Bitslice gate complexity).



function	CTCrypt 2025	CTCrypt 2026
I	16	15
ν_0	19	18
σ	19	15
α	11	
ω	5	
$\odot$	$30 \times 2 = 60$	
ν_1	11	
φ	16	
<i>Multiplexer</i>	12	
	169	163

Algorithm Битово-параллельная реализация I

Require: $\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3], \mathbf{x}[i] \in \mathbb{F}_2^w, i = 0, 1, 2, 3$

Ensure: $\mathbf{x}[5], \mathbf{x}[6], \mathbf{x}[7], \mathbf{x}[8] = I(\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3])$

- 1: $x[5] \leftarrow x[1] \oplus x[3]$
 - 2: $x[6] \leftarrow x[0] \oplus (x[5] \wedge x[2])$
 - 3: $x[8] \leftarrow x[3] \oplus (x[6] \wedge x[2])$
 - 4: $x[7] \leftarrow x[2] \oplus (x[5] \wedge x[8])$
 - 5: $x[6] \leftarrow x[6] \oplus x[7]$
 - 6: $x[5] \leftarrow x[5] \oplus (x[6] \wedge x[7])$
 - 7: $x[7] \leftarrow x[7] \oplus (x[6] \wedge x[8])$
 - 8: $x[8] \leftarrow x[8] \oplus (x[5] \wedge (x[6] \wedge x[7]))$
-

Algorithm Битово-параллельная реализация ν_0

Require: $\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3], \mathbf{x}[i] \in \mathbb{F}_2^w, i = 0, 1, 2, 3$

Ensure: $\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3] = \nu_0(\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3])$

- 1: $x[3] \leftarrow x[3] \oplus x[0]$
 - 2: $x[0] \leftarrow x[0] \oplus (x[3] \vee x[2])$
 - 3: $x[3] \leftarrow x[3] \oplus x[0]$
 - 4: $x[2] \leftarrow x[2] \oplus (x[1] \wedge x[0])$
 - 5: $x[0] \leftarrow \neg x[0]$
 - 6: $x[1] \leftarrow x[1] \oplus ((x[0] \vee x[3]) \wedge x[2])$
 - 7: $tmp \leftarrow x[2] \oplus x[3]$
 - 8: $x[3] \leftarrow x[3] \oplus (x[1] \vee tmp)$
 - 9: $x[2] \leftarrow x[1] \oplus x[0]$
 - 10: $x[1] \leftarrow x[0] \oplus x[3]$
 - 11: $x[0] \leftarrow tmp \oplus x[1]$
 - 12: $x[1] \leftarrow x[1] \oplus (x[2] \wedge x[0])$
-

Algorithm Битово-параллельная реализация σ

Require: $\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3], \mathbf{x}[i] \in \mathbb{F}_2^w, i = 0, 1, 2, 3$

Ensure: $\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3] = \sigma(\mathbf{x}[0], \mathbf{x}[1], \mathbf{x}[2], \mathbf{x}[3])$

- 1: $tmp \leftarrow x[3] \oplus x[1]$
 - 2: $x[3] \leftarrow x[1] \oplus x[0]$
 - 3: $x[1] \leftarrow tmp \oplus x[0]$
 - 4: $tmp \leftarrow \neg(x[2] \oplus (tmp \wedge x[3]))$
 - 5: $x[2] \leftarrow x[3] \oplus (tmp \vee x[0])$
 - 6: $x[0] \leftarrow x[0] \oplus (x[1] \vee x[2])$
 - 7: $x[3] \leftarrow x[1] \oplus (tmp \vee x[0])$
 - 8: $x[1] \leftarrow tmp \oplus (x[3] \wedge x[2])$
 - 9: $x[0] \leftarrow x[0] \oplus x[3]$
-

Алгоритм Карацубы

Алгоритм Карацубы Анатолия Алексеевича - метод быстрого умножения, позволяющий перемножать два n -значных числа. Изобретен в 1960 году. Общая методика алгоритма - «разделяй и властвуй»: задачу делят на непересекающиеся подзадачи того же типа, но меньшего размера, причём каждая подзадача вычисляется отдельно, а результаты решения в итоге объединяются в решение исходной задачи.

Алгоритм снижает умножение двух n -разрядных чисел до трёх умножений чисел вдвое меньшей длины. Для вычисления каждого из трёх умножений используется тот же метод рекурсивно.

Асимптотическая оценка сложности умножения по рекурсивному алгоритму Карацубы $O(n^{\log_2 3})$, где $\log_2 3 \approx 1,5849625$.

В режиме MGM выполняется умножение в полях $GF(2^{64})$ и $GF(2^{128})$.

Алгоритм Карацубы позволяет реализовать умножение двух многочленов степени $n - 1$. В нашем случае, $n \in \{64, 128\}$.

Умножение многочленов по алгоритму Карацубы

Пусть $n = 2m$ (если n не является степенью числа 2, дополняем нулевыми коэффициентами до ближайшей степени двойки для эффективного применения рекурсивной схемы Карацубы). Представим:

$$a(x) = a_H(x) \cdot x^m + a_L(x), \quad b(x) = b_H(x) \cdot x^m + b_L(x),$$

где $a_L, b_L \in \mathbb{F}_2^m$ — младшие m коэффициентов, $a_H, b_H \in \mathbb{F}_2^m$ — старшие, $\mathbb{F}_2 = \{0, 1\}$.

Тогда:

$$\begin{aligned} a(x) \cdot b(x) &= (a_H x^m + a_L) \cdot (b_H x^m + b_L) \\ &= (a_H \cdot b_H) x^{2m} \oplus (a_L \cdot b_L) \\ &\quad \oplus (a_H \cdot b_L \oplus a_L \cdot b_H) x^m. \end{aligned}$$

Суть оптимизации — выразить слагаемое $(a_H \cdot b_L \oplus a_L \cdot b_H)$ следующим образом:

$$a_H \cdot b_L \oplus a_L \cdot b_H = (a_H \oplus a_L) \cdot (b_H \oplus b_L) \oplus a_H \cdot b_H \oplus a_L \cdot b_L.$$

Приведение по модулю неприводимого многочлена

В основе bitslice реализации приведения многочлена по модулю неприводимого многочлена - алгоритм из работы⁵.

Результат умножения $a(x) = m1(x) \cdot m2(x)$, где $\deg(m1(x)) = n - 1$, $\deg(m2(x)) = n - 1$, можно представить $a(x) = c(x) \cdot x^n + b(x)$, $n \in \{64, 128\}$.

Обозначение: M^n - старшая, а L^n - младшая половина многочлена.

Например, $M^n(a(x)) = c(x)$, $L^n(a(x)) = b(x)$.

В работе⁵ из формулы (22) следует алгоритм вычисления остатка:

$$p(x) = (c(x) \cdot x^n) \bmod g(x) = L^n(g^*(x) \cdot M^n(c(x) \cdot q^+(x))), \deg(p(x)) = n - 1.$$

$$n = 128: g(x) = x^{128} + x^7 + x^2 + x + 1, g^*(x) = x^7 + x^2 + x + 1, q^+(x) = g(x),$$

$$n = 64: g(x) = x^{64} + x^4 + x^3 + x + 1, g^*(x) = x^4 + x^3 + x + 1, q^+(x) = g(x).$$

В итоге, приведение по модулю результата: $a(x) \bmod g(x) = p(x) \oplus b(x)$.

Т.к. $g^*(x)$ и $q^+(x)$ являются фиксированными многочленами, то вычисление $p(x)$ можно представить в виде линейного преобразования операциями XOR.

⁵Shay Guero, Michael E. Kounavis, Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode.

Сложность bitslice зашифрования «Магма», «Кузнечик»

В оценке сложности не учитываются инструкции присваивания = и транспонирование данных.

Зашифрование w 64-битных блоков «Магма», в реализации $w = 256$.

Функция	Количество инструкций
Сложение с раунд.ключом в кольце $\mathbb{Z}_{2^{32}}$	160
Подстановки $\pi_0, \dots, \pi_7$	132
Операция XOR с учетом $\lll 11$	32
Всего	$10\,368 = (160 + 132 + 32) \cdot 32$

Зашифрование w 128-битных блоков «Кузнечик»

Функция	Количество инструкций
$\oplus$ с раундовым ключом	128
Подстановка π	$2608 = 163 \cdot 16$
Линейное преобразование	$2768 = 173 \cdot 16$
Всего	$49\,664 = (128 + 2608 + 2768) \cdot 9 + 128$

В «Кузнечик» в 10-м раунде лишь сложение с раундовым ключом.

Сложность bitslice умножения многочленов и производительность зашифрования

Операция	$n = 64$	$n = 128$
Carry-less умножение (полное)	4 108	12 833
Приведение по модулю (полное)	196	445
Суммарная сложность $\otimes$	4 304	13 278

Блочный шифр	Зашифрование в режиме простой замены (МБайт/с)
«Кузнечик»	410
«Магма»	980

«Кузнечик» немного ускорился в сравнении с реализацией STCrypt 2025 за счёт представления расширенного ключа в виде bitslice векторов и применения инструкции **`_mm256_i32gather_epi32`** при считывании векторов из несмежных областей памяти перед транспонированием.

Сравнение производительности реализаций

«Кузнечик» MGM

Длина		Количество сообщений	Classic Реализация ⁶	Bitslice	
l_A	l_P			Hybrid	Fully
1 MB	1 MB	256	100 MB/sec	234 MB/sec	226 MB/sec
0 MB	1 MB	256	77	184	177
1 MB	0 MB	256	142	321	312
8 KB	8 KB	16384	101	226	227
0 KB	8 KB	16384	77	175	179
8 KB	0 KB	16384	139	298	309

1 MB - 1 Мегабайт (2^{20} байт), 1 KB - 1 Килобайт (2^{10} байт).

Производительность реализаций выражается в количестве Мегабайт данных, обрабатываемых за 1 секунду (MB/sec).

Эксперименты проведены на ЭВМ: процессор Intel Core i5 8400, ОС Ubuntu 24.04, компилятор GCC, опции компилятора -O2 -Wall -mavx2 -mpclmul. Все реализации однопоточные.

⁶Реализация «Кузнечик» MGM, МГТУ им.Баумана, <https://github.com/Roninsc2>.

Сравнение производительности реализаций

«Магма» MGM

Длина		Количество сообщений	Classic Реализация ⁷	Bitslice	
l_A	l_P			Hybrid	Fully
1 MB	1 MB	256	163 MB/sec	540 MB/sec	507 MB/sec
0 MB	1 MB	256	114	431	403
1 MB	0 MB	256	272	711	685
8 KB	8 KB	16384	158	517	515
0 KB	8 KB	16384	112	404	415
8 KB	0 KB	16384	246	655	680

⁷Авторская реализация «Магма» MGM с использованием AVX2. Умножение в поле использует инструкцию `_mm_clmulepi64_si128 (pclmulqdq)`.



<https://github.com/ASSol82/Kuznechik-MGM-bitslice>

<https://gitflic.ru/project/assolovye82/kuznechik-mgm-bitslice>



<https://github.com/ASSol82/Magma-MGM-bitslice>

<https://gitflic.ru/project/assolovye82/magma-mgm-bitslice>

