УДК 004.9 https://doi.org/10.51301/vest.su.2021.v143.i1.31

**Е.Н. Сейткулов*, Р.М. Оспанов, Б.Б.Ергалиева**
Евразийский национальный университет им. Л.Н. Гумилева, Нур-Султан, Казахстан
*e-mail: yerzhan.seitkulov@gmail.com

# ПРИМЕР КРИПТОГРАФИЧЕСКОЙ ХЕШ-ФУНКЦИИ, ПОСТРОЕННОЙ НА ОСНОВЕ МОДИФИЦИРОВАННОЙ СХЕМЫ SPONGE

**Аннотация.** В настоящее время схема «Sponge» является наиболее удачным и перспективным способом построения современных криптографических хэш-функций. Целью данной статьи является построение примера криптографической хеш-функции, основанной на этой схеме. Основным и важным компонентом схемы является внутренняя функция, являющейся преобразованием фиксированной длины или перестановкой, оперирующей с фиксированным числом битов, составляющих внутреннее состояние функции. Классическая схема «Sponge» и большинство ее модификаций предполагают в своем составе только одну внутреннюю функцию. В данной работе рассматривается модификация этой схемы, которая предполагает использование уже множества внутренних функций. Рассматриваются три новых варианта внутренней функции. Во-первых, рассматривается вариант внутренней функции, основанный на использовании словарных регистров сдвига с обратной связью по переносу кольцевой конфигурации. Во-вторых, рассматривается новый вариант внутренней функции, основанный на перестановке Keccak. В-третьих, рассматривается внутренняя функция, построенная с помощью обобщенной методологии проектирования AES. Далее на основе модифицированной схемы и используя эти три внутренние функции строится алгоритм хеширования. Выбор одной из этих трех внутренних функций в составе всей схемы определяется с помощью зависящих от сообщения битов выбора, сгенерированных псевдослучайным образом.

**Ключевые слова**: информационная безопасность, криптография, хеш-функция, схема «Sponge», внутренняя функция.

**Введение.** Проверка стойкости разрабатываемых криптографических хеш-функций ко всем существующим на текущий момент времени методам криптоанализа позволяет быть уверенным в их безопасности и их безопасном использовании, а также в их стандартизации. Однако по мере дальнейших исследований и усовершенствования криптоаналитических методов обнаруживаются слабые места в уже известных и широко используемых хеш-функциях, которые могут привести к частичному или полному нарушению их безопасности.

Например, в 2008 году в работах [1], [2] были представлены успешные теоретические криптоаналитические атаки на криптографическую хеш-функция, определенную в российском стандарте ГОСТ Р 34.11-94 «Информационная технология. Криптографическая защита информации. Функция хэширования». В 2005 году в работах [3], [4] была представлена теоретическая атака на криптографическую хеш-функцию SHA-1, определенную в стандарте FIPS PUB 180-1 Secure Hash Standard. Затем еще был опубликован целый ряд работ по теоретическому криптоанализу SHA-1. А в 2015 году в работе [5] была представлена первая практическая атака на внутреннюю функцию сжатия SHA-1. В 2017 году была осуществлена практическая атака на SHA-1 [6], в результате которой были получены два разных PDF-файла с одним и тем же хеш-значением SHA-1. Позже в еще ряде работ [7], [8], [9] были опубликованы новые практические атаки на SHA-1.

Еще одна проблема, которая может поколебать уверенность в безопасном использовании уже разработанных криптографических хеш-функций (и даже стандартизированных), заключается в том, что проверка на стойкость к криптоанализу может не обнаружить бэкдоры, которые возможно, были встроены в хеш-функции как на этапе их программной и/или аппаратной реализации, так и на этапе математического проектирования. Сами бэкдоры могут быть сконструированы с помощью сложных криптографических методов, которые затрудняют их обнаружение. Знание бэкдора позволяет осуществлять эффективную

практическую атаку на хеш-функцию.

Например, в работе [10] представлена версия SHA-1, содержащая бэкдор, с помощью которой можно успешно находить коллизии с приблизительной сложностью в 248 вычислений. Для сравнения, считается, что вычислительная сложность нахождения коллизии для стандартного алгоритма SHA-1 составляет более 263 вычислений. В работе [11] представлена версия с бэкдором алгоритма BLAKE (финалиста конкурса SHA-3), для которой также можно успешно находить коллизии. Также варианты с бэкдорами были исследованы для известных криптографических хеш-функций Стрибог (российский стандарт алгоритма хеширования) и Keccak (победитель конкурса SHA-3) в работах [12], [13] соответственно. Известный случай [14] существования возможного бэкдора в стандартизированном криптографическом алгоритме генерации псевдослучайных чисел Dual_EC_DRBG [15], или недавнее исследование S-блока [16], используемого в Стрибог (и Кузнечик (российский стандарт алгоритма блочного шифрования) [17]), вызывают подозрения, что и для существующих стандартизированных криптографических хеш-функций нельзя исключать возможность наличия бэкдоров.

В настоящее время в Казахстане в качестве стандартов криптографической хэш-функции используются иностранные алгоритмы, и таким образом, в связи с вышеизложенным, существует определенная необходимость в разработке собственной криптографической хеш-функции.

Задача разработки нового криптографического алгоритма хеширования является сложной и противоречивой. Разрабатываемый алгоритм должен удовлетворять основным требованиям безопасности, предъявляемым криптографическим хеш-функциям, а также обладать высокими показателями быстродействия. Внутренняя структура алгоритма должна быть простой, ясной, обоснованной и обеспечивать возможность эффективной реализации на программном и/или аппаратном уровнях. В данной работе для решения этой задачи предлагается использовать хорошо исследованные конструкции, обоснованно гарантирующие соответствие требованиям безопасности.

Наиболее популярной и перспективной конструкцией является схема «Sponge» («криптографическая губка») [18], [19]. С её помощью кроме хеш-функций можно создавать такие криптопримитивы, как блочные симметричные шифры, коды аутентификации сообщения и поточные шифры. Более того, по этой схеме был спроектирован алгоритм Keccak [20], ставший победителем конкурса SHA-3. Схему можно описать как последовательность следующих основных преобразований, в результате которых вычисляется хеш-значение заданного сообщения:

1) Дополнение (padding), при котором входное сообщение дополняется некоторым количеством битов так, чтобы длина дополненного сообщения была кратна заданной длине блока сообщения.

2) Инициализация состояния, при котором задается некоторое начальное значение состояния.

3) «Фаза впитывания» (absorbing phase), при котором сообщение сжимается итеративно.

4) «Фаза выжимания» (squeezing phase), при котором в результате требуемое хеш-значение сообщения извлекается.

Существующие различные модификации схемы отличаются друг от друга различными способами дополнения, вариантами реализации инициализации состояния. Но основным и важным компонентом схемы «Sponge» является внутренняя функция, являющейся преобразованием фиксированной длины или перестановкой, оперирующей с фиксированным числом битов, составляющих внутреннее состояние функции.

Классическая схема «Sponge» и большинство ее модификаций предполагают в своем составе только одну внутреннюю функцию. В данной работе рассматривается пример алгоритма хеширования на основе модифицированной схемы «Sponge», которая предполагает использование уже множества внутренних функций. В рассматриваемом алгоритме используются три различные внутренние функции.

**Методы.** Для построения криптографической хеш-функции применим вариант модификации схемы «Sponge», который предполагает использование множества внутренних функций [21], [22]. Согласно этой схеме над входным сообщением выполняются следующие преобразования:

1) К входному сообщению применяется функция дополнения, в результате которой к сообщению добавляются дополнительные биты, и сообщение представляется в виде конкатенации блоков определенной одинаковой длины r.

2) Выполняется функция инициализации, т.е. заполнение начального значения некоторой переменной S, битовой последовательности некоторой определенной длины, называемой состоянием.

3) К состоянию S и первому блоку сообщения применяется функция инжекции сообщения, в результате которой преобразовывается значение состояния S.

4) Из заданного множества внутренних функций выбирается функция f с помощью функции выбора.

5) К состоянию S применяем функцию f.

6) К состоянию S и следующему блоку сообщения применяется функция инжекции сообщения, в результате которой преобразовывается значение состояния S.

7) Из заданного множества внутренних функций выбирается функция f с помощью функции выбора.

8) К состоянию S применяем функцию f.

9) Повторяются шаги 6, 7 и 8 до тех пор, пока не будут обработаны таким образом все блоки сообщения.

10) Из состояния S получаем первый выходной блок длиной r' (не обязательно равной r) с помощью функции извлечения.

11) Из заданного множества внутренних функций выбирается функция f с помощью функции выбора.

12) К состоянию S применяем функцию f.

13) Из состояния S получаем следующий выходной блок длиной r' с помощью функции извлечения.

14) Из заданного множества внутренних функций выбирается функция f с помощью функции выбора.

15) К состоянию S применяем функцию f.

16) Повторяются шаги 13, 14 и 15 до тех пор, пока общая длина всех возвращенных блоков не будет больше или равна требуемой длине хеш-значения.

17) К полученным выходным блокам применяется функция завершения, в результате которой получается требуемое хеш-значение.

В качестве внутренних функций применим следующие три преобразования.

А) Внутренняя функция $F_0$ строится на основе обобщенной методологии проектирования AES с размерностью 9 [23].

Внутренняя функция $F_0$ итеративно использует S-блоки, линейное преобразование и перестановку.

Внутренняя функция F0 использует два 4-битовых S-блока $S_0$ и $S_1$, заданные следующими таблицами.

$S_0$

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_0(x)$ | 9 | 0 | 4 | B | D | C | 3 | F | 1 | A | 2 | 6 | 7 | 5 | 8 | E |

$S_1$

| x | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S_1(x)$ | 3 | C | 6 | D | 5 | 7 | 1 | 9 | F | 2 | 0 | 4 | B | A | E | 8 |

Внутренняя функция $F_0$ использует линейное преобразование L, реализующее (4, 2, 3) МДР код над $GF(2^4)$, где умножение определяется как умножение двоичных многочленов по модулю неприводимого многочлена $x^4 + x + 1$. L преобразовывает пару 4-битовых слов $X = (x_0, x_1, x_2, x_3)$, $Y = (y_0, y_1, y_2, y_3)$ следующим образом:

$L(x_0, x_1, x_2, x_3, y_0, y_1, y_2, y_3) = (x_0 + (y_1 + x_2), x_1 + (y_2 + x_3 + x_0), x_2 + (y_3 + x_0) + (y_0 + x_1), x_3 + (y_0 + x_1), y_0 + x_1, y_1 + x_2, y_2 + x_3 + x_0, y_3 + x_0)$.

Внутренняя функция $F_0$ использует перестановку P. P – перестановка 512 элементов, являющаяся композицией трех перестановок 512 элементов $P = p_0 \circ p_1 \circ p_2$.

$p_0$ – перестановка 512 элементов, определяемая следующим образом: $p_0(x_i) = x_i$, $i = 0, \ldots, 255$, $p_0(x_{2i+0}) = x_{2i+1}$, $i = 128, \ldots, 255$, $p_0(x_{2i+1}) = x_{2i+0}$, $i = 128, \ldots, 255$.

$p_1$ – перестановка 512 элементов, определяемая следующим образом: $p_1(x_i) = x_{2i}$, $i = 0, \ldots, 255$, $p_1(x_{i+256}) = x_{2i+1}$, $i = 0, \ldots, 255$.

$p_2$ – перестановка 512 элементов, определяемая следующим образом: $p_2(x_{4i+0}) = x_{4i+0}$, $i = 0, \ldots, 127$, $p_2(x_{4i+1}) = x_{4i+1}$, $i = 0, \ldots, 127$, $p_2(x_{4i+2}) = x_{4i+3}$, $i = 0, \ldots, 127$, $p_2(x_{4i+3}) = x_{4i+2}$, $i = 0, \ldots, 127$.

Внутренняя функция $F_0$ использует раундовое преобразование R, выполняемое над 2048 битовыми словами. Раундовое преобразование R(A, C) определяется на основе определенных выше S-блоков $S_0$ и $S_1$, линейного преобразования L и перестановки P следующим образом.

Пусть $A = (a_0 \| a_1 \| \ldots \| a_{511})$ – 2048 битовое слово, где $a_i$ ($i = 0, \ldots, 511$) - 4-битовые слова.

Пусть $C = (C_0 \| C_1 \| \ldots \| C_{511})$ – 512 битовое слово.

1) К слову A применяется S-блоки $S_0$ и $S_1$. Для каждого $i = 0, \ldots, 511$, $a_i$ заменяется на $a'_i = S_0(a_i)$, если $C_i = 0$, или $a_i$ заменяется на $a'_i = S_1(a_i)$, если $C_i = 1$.
2) К полученному слову $A' = (a'_0 \| a'_1 \| \ldots \| a'_{511})$ применяется линейное преобразование L. Для каждого $i = 0, \ldots, 255$, пара 4-битовых слов $(a'_{2i}, a'_{2i+1})$ заменяется на $(a''_{2i}, a''_{2i+1}) = L(a'_{2i}, a'_{2i+1})$.
3) К полученному слову $A'' = (a''_0 \| a''_1 \| \ldots \| a''_{511})$ применяется перестановка P.

Внутренняя функция F0 использует в качестве раундовых констант 512 битовые слова $C_r$, $r = 0, 1, \ldots, 47$, определяемые следующим образом.

$C_0$ – целая часть числа $\left(\sqrt{2} - 1\right) \times 2^{512}$.

Для генерации остальных констант используются преобразование $R_0$ и перестановка $P_0$.

$P_0$ – перестановка 128 элементов, аналогичная перестановке P, является композицией трех перестановок 128 элементов $P_0 = p'_0 \circ p'_1 \circ p'_2$.

$p'_0$ – перестановка 128 элементов, определяемая следующим образом: $p'_0(x_i) = x_i$, $i = 0, \ldots, 63$, $p'_0(x_{2i+0}) = x_{2i+1}$, $i = 32, \ldots, 63$, $p'_0(x_{2i+1}) = x_{2i+0}$, $i = 32, \ldots, 63$.

$p'_1$ – перестановка 128 элементов, определяемая следующим образом: $p'_1(x_i) = x_{2i}$, $i = 0, \ldots, 63$, $p'_1(x_{i+64}) = x_{2i+1}$, $i = 0, \ldots, 63$.

$p'_2$ – перестановка 128 элементов, определяемая следующим образом: $p'_2(x_{4i+0}) = x_{4i+0}$, $i = 0, \ldots, 31$, $p'_2(x_{4i+1}) = x_{4i+1}$, $i = 0, \ldots, 31$, $p'_2(x_{4i+2}) = x_{4i+3}$, $i = 0, \ldots, 31$, $p'_2(x_{4i+3}) = x_{4i+2}$, $i = 0, \ldots, 31$.

$R_0$ – преобразование, выполняемое над 512 битовыми словами, аналогичное преобразованию R. $R_0$ определяется следующим образом.

Пусть $A = (a_0 \| a_1 \| \ldots \| a_{127})$ – 512 битовое слово, где $a_i$ ($i = 0, \ldots, 127$) - 4-битовые слова.

1) К слову A применяется S-блок $S_0$. Для каждого $i = 0, \ldots, 127$, $a_i$ заменяется на $a'_i = S_0(a_i)$.
2) К полученному слову $A' = (a'_0 \| a'_1 \| \ldots \| a'_{127})$ применяется линейное преобразование L. Для каждого $i = 0, \ldots, 63$, пара 4-битовых слов $(a'_{2i}, a'_{2i+1})$ заменяется на $(a''_{2i}, a''_{2i+1}) = L(a'_{2i}, a'_{2i+1})$.
3) К полученному слову $A'' = (a''_0 \| a''_1 \| \ldots \| a''_{127})$ применяется перестановка $P_0$.

Константы $C_r = R_0(C_{r-1})$, $r = 1, 2, \ldots, 47$.

Внутренняя функция $F_0$ выполняет над входными битами следующие преобразования.

1) Входные 2048 битов $A = (a_0 \| a_1 \| \ldots \| a_{2047})$ группируются по 4 бита. Получается 512 4-битовых слов (512 полубайтов) $Q_0 = (q_{0\,0} \| q_{0\,1} \| \ldots \| q_{0\,511})$, где

$q_{0\,2i} = a_i \,||\, a_{i+512} \,||\, a_{i+1024} \,||\, a_{i+1536}$, i = 0, …, 255,

$q_{0\,2i+1} = a_{i+256} \,||\, a_{i+768} \,||\, a_{i+1280} \,||\, a_{i+1792}$, i = 0, …, 255.

2) Далее 48 раз применяется раундовое преобразование R с раундовыми константами $C_r$, получая в результате $Q_{r+1} = R(Q_r, C_r)$, r = 0, 1, …, 47.

3) Полученные в результате последнего раунда 512 4-битовых слов (512 полубайтов) $Q_{48} = (q_{48\,0} \,||\, q_{48\,1} \,||\, \dots \,||\, q_{48\,511})$ разгруппировываются. Получаются выходные 2048 битов $B = (b_0 \,||\, b_1 \,||\, \dots \,||\, b_{2047})$, где

$b_i \,||\, b_{i+512} \,||\, b_{i+1024} \,||\, b_{i+1536} = q_{48\,2i}$, i = 0, …, 255,

$b_{i+256} \,||\, b_{i+768} \,||\, b_{i+1280} \,||\, b_{i+1792} = q_{48\,2i+1}$, i = 0, …, 255.

Б) Внутренняя функция $F_1$ основана на словарных регистрах сдвига с обратной связью по переносу кольцевой конфигурации [24], [25].

Словарный регистр сдвига с обратной связью по переносу кольцевой конфигурации состоит из:

1) основного сдвигового регистра;

2) функции обратной связи;

3) регистра переноса.

Сдвиговый регистр – это конечная последовательность w ячеек $m = (m_0, m_1, ..., m_{w-1})$, где каждая ячейка предназначена для хранения одного r-битового слова.

Регистр переноса – также конечная последовательность w ячеек $c = (c_0, c_1, ..., c_{w-1})$, где каждая ячейка предназначена для хранения одного r-битового слова.

Функция обратной связи определяется следующими соотношениями:

$m(t+1) = T \cdot m(t) + c(t) \bmod 2$, $c(t+1) = T \cdot m(t) + c(t) \div 2$, где T – матрица перехода.

Матрица перехода T – квадратная блочная матрица порядка w с элементами $T(i, j)$, определяемыми следующим образом:

1) для i = 0, …, w-2 $T(i, i+1)$ – единичная матрица порядка r,

2) $T(w-1, 0)$ – единичная матрица порядка r,

3) для некоторых $i = i_0, ..., i_{f-1}$, где $f \le w-1$,

$T(i, j_i)$ – матрица порядка r, представляющая сдвиг вправо или влево или циклический сдвиг вправо или влево на некоторое s количество битов,

4) для остальных i и j $T(i, j)$ – нулевая матрица порядка r.

В общем случае матрица перехода имеет следующий вид:

$$T = \begin{pmatrix} I_r & & R_0 & & \\ R_1 & I_r & & (0) & \\ & & I_r & & R_2 \\ & (0) & & & \\ & & R_{w-2} & & I_r \\ I_r & R_{w-1} & & & \end{pmatrix},$$

где

$I_r$ - единичная матрица порядка r,

$R_i$ - матрица порядка r, представляющая сдвиг вправо или влево или циклический сдвиг вправо или влево на некоторое s количество битов.

Матрица, представляющая сдвиг влево, может быть представлена следующей матрицей SL для r-битного слова:

$SL \cdot (x_0, \ldots, x_{r-1})^t = (x_1, \ldots, x_{r-1}, 0)^t$.

Матрица, представляющая сдвиг вправо, может быть представлена следующей матрицей SR для r-битного слова:

$SR \cdot (x_0, \ldots, x_{r-1})^t = (0, x_0, x_1, \ldots, x_{r-2})^t$.

Матрица, представляющая циклический сдвиг влево, может быть представлена следующей матрицей RL для r-битного слова:

$RL \cdot (x_0, \ldots, x_{r-1})^t = (x_{r-1}, x_0, x_1, \ldots, x_{r-2})^t$.

Матрица, представляющая циклический сдвиг вправо, может быть представлена следующей матрицей RR для r-битного слова:

$RR \cdot (x_0, \ldots, x_{r-1})^t = (x_1, \ldots, x_{r-1}, x_0)^t$.

В этом случае параметры $R_i$ матрицы T равны $SL^a$ , $SR^b$ , $RL^c$ или $RR^d$ , где a, b, c и d представляют собой искомые сдвиги или циклические сдвиги.

Матрица перехода должна удовлетворять следующим свойствам:

1) $\log_2(q) \geq b$, $det(T) \neq 0$,

2) $q = det(I - 2T)$ - простое число и порядок 2 по модулю q равен $|q| - 1$.

Алгоритм выбора матрицы перехода [26]:

1) Заполняется матрица $T = (T(i, j))$, $0 \leq i, j < w$, следующим образом:

а) для i = 0, …, w-2 $T(i, i+1)$ – единичная матрица порядка r,

б) $T(w-1, 0)$ – единичная матрица порядка r,

в) для остальных j и j $T(i, j)$ – нулевая матрица порядка r.

2) Из номеров 0, …, w-1 выбираем случайным образом номера $i_0, \ldots, i_{f-1}$; $j_0, \ldots, j_{f-1}$. Из чисел -r/2, …,-1, 1, …, r/2 выбираем случайным образом числа $s_0, \ldots, s_{f-1}$.

3) Цикл по l от 0 до f-1:

$T(i_l, j_l) := T(i_l, j_l) + SL^{s_l}$, если $s_l > 0$,

$T(i_l, j_l) := T(i_l, j_l) + SR^{-s_l}$, если $s_l < 0$,

где $SL^{s_l}$ - матрица порядка r, представляющая сдвиг влево на $s_l$ количество битов, $SR^{-s_l}$ - матрица порядка r, представляющая сдвиг вправо на $s_l$ количество битов.

Вычисляется число q = det(I – 2T).

Если q простое и порядок 2 по модулю q равен $|q| - 1$, то в качестве результата алгоритма возвращается матрица T, в противном случае возвращаемся к шагу 2).

Внутренняя функция $F_1$, основанная на словарных регистрах сдвига с обратной связью по переносу кольцевой конфигурации, определяется следующим образом.

Функция выполняет над входными битами следующие преобразования:

Входные биты определяют начальное значение сдвигового регистра:

$m_0(0), m_1(0), \ldots, m_{w-1}(0)$,

где $m_i(0)$ – r-битовое слово.

Начальное состояние регистра переноса:

$c_0(0), c_1(0), \ldots, c_{w-1}(0)$,

где $c_i(0)$ – r-битовое нулевое слово.

Следующие значения состояний сдвигового регистра и регистра переноса определяются следующим образом.

Для некоторых $i = i_0, \ldots, i_{f-1}$ , где $f \leq w-1$, определенных в результате генерации матрицы перехода, значения состояний сдвигового регистра

$$m_i(t) = m_{i+1}(t-1) + \left(m_{j_i}(t-1) \lll s_{j_i}\right) + c_i(t-1),$$

и регистра переноса

$$c_i(t) = m_{i+1}(t-1) \cdot \left(m_{j_i}(t-1) \lll s_{j_i}\right) + m_{i+1}(t-1) \cdot c_i(t-1) + \left(m_{j_i}(t-1) \lll s_{j_i}\right) \cdot c_i(t-1).$$

Далее

$$m_{w-1}(t) = m_0(t-1) + \left(m_{j_{w-1}}(t-1) \lll s_{j_{w-1}}\right) + c_{w-1}(t-1),$$

$$c_{w-1}(t) = m_0(t-1) \cdot \left(m_{j_{w-1}}(t-1) \lll s_{j_{w-1}}\right) + m_0(t-1) \cdot c_{w-1}(t-1) + \left(m_{j_{w-1}}(t-1) \lll s_{j_{w-1}}\right) \cdot c_{w-1}(t-1).$$

Для остальных i значения состояний сдвигового регистра определяются следующим образом;

$$m_i(t) = m_{i+1}(t-1).$$

Здесь $\lll s$ обозначает сдвиг влево на s битов или сдвиг вправо на $|s|$, если $s \le 0$, + - побитовое сложение (XOR), $\cdot$ - умножение по модулю $2^r$.

Выполняется итерация D+4 раза, где D – диаметр графа, матрицей смежности которого является матрица T.

Далее выполняется w итераций. В конце каждой итерации выполняется побайтовые замены с помощью таблицы замен (S-блок) в последовательности слов основного регистра $m_{i_0} \lll s_{i_0}, ..., m_{i_{f-1}} \lll s_{i_{f-1}}$ для некоторых для некоторых $i = i_0, ..., i_{f-1}$, где $f \le w-1$, определенных в результате генерации матрицы перехода.

Затем к преобразованным словам применяется побитовое сложение (XOR), получая в результате r-битовое слово. В конце получается всех w итераций получается b = wr выходных битов.

В) Внутренняя функция $F_2$ строится аналогично перестановке Keccak, но имеет ряд основных отличий.

Внутренняя функция $F_2$ представляет собой перестановку, которую можно описать как последовательность операций над 2048-битовым состоянием S. Состояние S разбивается на 32 64-битовых слова $S_0$, $S_1$, …, $S_{31}$. Слова записываются в матрицу размера 4 × 8 последовательно слева направо и сверху вниз. Четверки $S_i$, $S_{i+8}$, $S_{i+16}$, $S_{i+24}$, i = 0, 1, ⋯ , 7, называются вертикальными плоскостями, а восьмерки $S_{8j}$, $S_{8j+1}$, ⋯ , $S_{8j+7}$, j = 0, 1, 2, 3 - горизонтальными. Другими словами, состояние S можно рассматривать как трехмерный битовый массив S(x, y, z), x = 0, 1, …, 3, y = 0, 1, …, 7, z = 0, 1, …, 63.

Действие внутренней функции состоит в 32-кратном повторении последовательности следующих преобразований.

1) $f_1(S(x, y, z)) = S(x, y, z) + S(0, y-1, z) + S(1, y-1, z) + S(2, y-1, z) + S(3, y-1, z) + S(0, y+1, z-1) + S(1, y+1, z-1) + S(2, y+1, z-1) + S(3, y+1, z-1)$ при x = 0, 1, …, 3, y = 1, …, 6, z = 1, …, 63;

$f_1(S(x, 0, 0)) = S(x, 0, 0) + S(0, 7, 0) + S(1, 7, 0) + S(2, 7, 0) + S(3, 7, 0) + S(0, 1, 63) + S(1, 1, 63) + S(2, 1, 63) + S(3, 1, 63)$ при x = 0, 1, …, 3;

$f_1(S(x, 7, 0)) = S(x, 7, 0) + S(0, 6, 0) + S(1, 6, 0) + S(2, 6, 0) + S(3, 6, 0) + S(0, 0, 63) + S(1, 0, 63) + S(2, 0, 63) + S(3, 0, 63)$ при x = 0, 1, …, 3.

2) $f_2(S(x, y, z)) = S(x, y, z - (t + 1)(t + 2)/2)$,

где t такое, что $0 \le t < 32$ и $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ в $GF(4)^{2\times 2}$, или t = -1 при x = y =0.

3) $f_3(S(x, y, z)) = S(x', y', z)$, где x', y' такие, что $\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$.

4) $f_4$ – оптимальный 8 битовый S-блок, применяемый к 8-битовым строкам S(x, 0, z),

S(x, 1, z), S(x, 2, z), S(x, 3, z), S(x, 4, z), S(x, 5, z), S(x, 6, z), S(x, 7, z).

В данной работе этот S-блок не определяется. Можно использовать уже существующие оптимальные S-блок или сформировать с помощью известных методов генерации оптимальных S-блоков.

5) $f_5$ состоит в сложении слова $S_{31}$ с 64-битовой константой C. Константа C вычисляется при помощи словарного регистра сдвига с обратной связью по переносу кольцевой конфигурации.

**Результат.** Далее рассмотрим пример алгоритма хеширования на основе модифицированной схемы «Sponge», в котором задается множество трех внутренних функций, указанных выше. Согласно этому алгоритму над входным сообщением выполняются следующий преобразования.

1) К входному сообщению применяется функция дополнения, в результате которой к сообщению добавляются дополнительные биты, и сообщение представляется в виде конкатенации блоков определенной одинаковой длины r.

Функция дополнения определяется следующим образом. К входному сообщению (со стороны наименьшего значащего бита) добавляется один бит 1, за которым следует минимальное число битов 0 и в конце один бит 1, так что длина результата кратна длине блока. Причем к сообщению добавляется не менее 2 битов и не более количества битов, равного длине блока плюс один.

2) Выполняется функция инициализации, т.е. заполнение начального значения некоторой переменной S, битовой последовательности некоторой определенной длины b, называемой состоянием.

Функция инициализации определяется следующим образом. Начальное значение состояния – это b - 80 битов 0, затем 80-битовое представление слова “ТАҢБА” в кодировке ЮНИКОД (0...0 0000 0100 0010 0010 0000 0100 0001 0000 0000 0100 1010 0010 0000 0100 0001 0001 0000 0100 0001 0000)

3) К состоянию S и первому блоку сообщения применяется функция инжекции сообщения, в результате которой преобразовывается значение состояния S.

Функция инжекции сообщения определяется следующим образом. Выполняется побитово операция XOR между первыми r битами состояния S и первым блоком. И затем первые r бит состояния S заменяем на результат этой операции.

4) Из заданного множества внутренних функций выбирается функция $F_i$ с помощью функции выбора.

Функция выбора определяется следующим образом.

Входное сообщение делится на две части. Если длина m сообщения является четным числом, то делится на две части одинаковой длины, а если не четным, то две части с длинами [m/2]+1 и [m/2] .

Генерируется первый бит выбора. Для этого к первой части входного сообщения применяют корректор фон Неймана, т.е. биты входного сообщения рассматриваются парами: если в паре два одинаковых значения, то пара отбрасывается, если биты разные, то вместо пары записывается только первый бит в этой паре. Затем к результату применяют XOR корректор, т.е. все биты получившейся последовательности складываются по модулю 2. В результате получается первый бит выбора. Аналогичным образом генерируется второй бит выбора, применяя корректор фон Неймана и XOR корректор к второй части входного сообщения.

Первые r бит состояния S последовательно подаются на информационный вход первого 1-2 демультиплексора, на адресный вход которого подается первый бит выбора, а остальные c бит состояния S последовательно подаются на информационный вход второго 1-2 демультиплексора, на адресный вход которого также подается первый бит выбора. Если бит выбора - 0, то полученные на выходе биты первого 1-2 демультиплексора подаются на информационный вход третьего 1-2 демультиплексора, на адресный вход которого подается

второй бит выбора, а полученные на выходе второго 1-2 демультиплексора подаются на информационный вход четвертого 1-2 демультиплексора, на адресный вход которого также подается второй бит выбора. Далее к полученным на выходе r+c битам будет применена внутренняя функция F0, если второй бит выбора - 0, или будет применена внутренняя функция F1, если второй бит выбора - 1. Если же первый бит выбора – 1, то к полученным на выходе r+c битам будет применена внутренняя функция F2. Биты выбора инвертируются.

5) К состоянию S применяется выбранная внутренняя функция $F_i$, i=0,1,2.

6) К состоянию S и следующему блоку сообщения применяется функция инжекции сообщения, в результате которой преобразовывается значение состояния S.

7) Из заданного множества внутренних функций выбирается функция Fi с помощью функции выбора.

8) К состоянию S применяется выбранная внутренняя функция $F_i$, i=0,1,2.

9) Повторяются шаги 6, 7 и 8 до тех пор, пока не будут обработаны таким образом все блоки сообщения.

10) Из состояния S получаем первый выходной блок длиной r' (не обязательно равной r) с помощью функции извлечения.

Функция извлечения определяется следующим образом. Первые r' бит состояния S образуют выходной блок.

11) Из заданного множества внутренних функций выбирается функция $F_i$ с помощью функции выбора.

12) К состоянию S применяется выбранная внутренняя функция $F_i$, i=0,1,2.

13) Из состояния S получаем следующий выходной блок длиной r' с помощью функции извлечения.

14) Из заданного множества внутренних функций выбирается функция $F_i$ с помощью функции выбора.

15) К состоянию S применяется выбранная внутренняя функция $F_i$, i=0,1,2.

16) Повторяются шаги 13, 14 и 15 до тех пор, пока общая длина всех возвращенных блоков не будет больше или равна требуемой длине хеш-значения.

17) К полученным выходным блокам применяется функция завершения, в результате которой получается требуемое хеш-значение.

Функция завершения определяется следующим образом. Первые n бит полученной последовательности образуют хеш-значение.

**Обсуждение.** В данной статье рассмотрен новый вариант криптографической хеш-функции, построенный на основе модифицированной схемы «Sponge». Согласно этой схеме из заданного множества внутренних функций выбирается функция f с помощью функции выбора. В построенном алгоритме используется множество трех различных внутренних функций. Первая внутренняя функция строится на основе обобщенной методологии проектирования AES. Эта методология позволяет легко проектировать блочные шифры для зашифровывания больших блоков открытого текста с помощью небольших компонентов, представляя обрабатываемые данные в виде многомерных массивов. Внутренняя функция является блочным шифром, который обрабатывает 2048 битов, представляемых в виде 9-мерного массива из 512 4-битовых элементов размера 2×2×2×2×2×2×2×2×2. Каждый раунд шифрования состоит из трех преобразований (S-блоки, линейное преобразование и перестановка), аналогичных трем раундовым преобразованиям AES SubBytes, MixColumns и ShiftRows. Вторая внутренняя функция основана на использовании словарных регистров сдвига с обратной связью по переносу кольцевой конфигурации. Третья внутренняя функция аналогична перестановке Keccak, но обладает несколькими основными отличиями. Внутренняя функция оперирует над 2048-битовым состоянием S, который можно рассматривать как трехмерный битовый массив размера 4 × 8 × 64. В этой внутренней функции вместо 5-битового S-блока используется 8-битовый. В связи с этим изменены параметры трехмерного представления состояния. Для формирования раундовых констант вместо

регистра сдвига с линейной обратной связью используется словарный регистр сдвига с обратной связью по переносу кольцевой конфигурации.

**Источник финансирования.** Данная работа выполнена при финансовой поддержке грантового финансирования МЦРИАП, № AP06851124.

**ЛИТЕРАТУРА**

[1] Mendel F., Pramstaller N., Rechberger C. A (second) preimage attack on the GOST hash function. In: Nyberg K. (eds) Fast Software Encryption. FSE 2008. LNCS, vol. 5086, Springer, Berlin, Heidelberg, 2008. - pp. 224–234.

[2] Mendel F., Pramstaller N., Rechberger C., Kontak M., Szmidt J. Cryptanalysis of the GOST hash function. In: Wagner D. (eds) Advances in Cryptology – CRYPTO 2008. CRYPTO 2008. LNCS, vol. 5157, Springer, Berlin, Heidelberg, 2008. - pp. 162–178.

[3] Wang X., Yao A.C., Yao F. Cryptanalysis on SHA-1. In: Proc. of NIST Cryptographic Hash Workshop (2005). October 31 - November 1, 2005. http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf.

[4] Wang X., Yin Y.L. Yu H. Finding collisions in the full SHA-1. In: Shoup V. (eds) Advances in Cryptology – CRYPTO 2005. CRYPTO 2005. LNCS, vol. 3621, Springer, Berlin, Heidelberg, 2005. - pp. 17-36.

[5] Stevens M., Karpman P., Peyrin T. Freestart collision for full SHA-1. IACR Cryptology ePrint Archive, Report 2015/967, 2015. https://eprint.iacr.org/2015/967.

[6] Stevens M., Bursztein E., Karpman P., Albertini A., Markov Y. The First Collision for Full SHA-1. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, LNCS, vol. 10401, Springer, Cham, 2017. - pp.570–596.

[7] Stevens M. Attacks on Hash Functions and Applications. PhD thesis, Mathematical Institute, Faculty of Science, Leiden University, 2012.

[8] Leurent G., Peyrin T. From Collisions to Chosen-Prefix Collisions Application to Full SHA-1. In: Ishai Y., Rijmen V. (eds) Advances in Cryptology – EUROCRYPT 2019. LNCS, vol. 11478, Springer, Cham, 2019. - pp. 527–555.

[9] Leurent G., Peyrin T. SHA-1 is a Shambles First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust. IACR Cryptology ePrint Archive, Report 2020/014, 2020. https://eprint.iacr.org/2020/014.

[10] Albertini A., Aumasson J.-P., Eichlseder M., Mendel F., Schläffer M. Malicious hashing: Eve's variant of SHA-1. In: Joux, A., Youssef, A. (eds.) Selected Areas in Cryptography -- SAC 2014. SAC 2014. LNCS, vol. 8781, Springer, Cham, 2014. - pp. 1–19.

[11] Aumasson J.-P. Eve's SHA3 candidate: malicious hashing. (2011), https://www.aumasson.jp/data/papers/Aum11a.pdf.

[12] AlTawy R., Youssef A.M. Watch your constants: malicious streebog. IET Information Security, 9(6), 2015. - pp. 328–333.

[13] Morawiecki P. Malicious Keccak. IACR Cryptology ePrint Archive, Report 2015/1085, 2015. https://eprint.iacr.org/2015/1085.

[14] Shumow D., Ferguson N. On the possibility of a back door in the NIST SP800-90 Dual Ec Prng. In: CRYPTO 2007 Rump Session, 2007. http://rump2007.cr.yp.to/15- shumow.pdf.

[15] Barker E., Kelsey J. Recommendation for random number generation using deterministic random bit generators (revised). NIST Special Publication 800-90, 2007.

[16] Perrin L. Partitions in the S-Box of Streebog and Kuznyechik. IACR Transactions on Symmetric Cryptology, 2019(1). - pp. 302–329.

[17] National standard of the Russian Federation GOST R 34.12-2015 "Information Technology. Cryptographic data security. Block ciphers", English version, Standartinform, Moscow, 2015.

[18] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: Ecrypt Hash Workshop 2007 (May 2007); available as public comment to NIST, http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html

[19] Bertoni G., Daemen J, Peeters M., Van Assche G. Cryptographic sponge functions. Version 0.1, January 14, 2011. https://keccak.team/files/CSF-0.1.pdf.

[20] Bertoni G., Daemen J, Peeters M., Van Assche G. The Keccak reference. SHA-3 competition (round 3), 2011. https://keccak.team/sponge_duplex.html.

[21] Оспанов Р.М., Сейткулов Е.Н., Арапов Н.К., Ергалиева Б.Б. Модификация схемы построения криптографических хеш-функций SPONGE. Вестник КазНИТУ. - 2020. - N 5 (141). - С.520-525

[22] Оспанов Р.М., Сейткулов Е.Н. Киберщит: О различных реализациях схемы построения криптографических хэш-функций «Sponge». Материалы Международной научно-практической Web-конференции «Военно-техническое обеспечение деятельности вооруженных сил: мировой опыт и тенденции развития», 20-22 июля 2020 года. - Нур-Султан: Из-во НУО, 2020. - С.305-308

[23] Wu H. The Hash Function JH. 2011. https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf.

[24] Оспанов Р.М., Сейткулов Е.Н. О способах проектирования внутренней функции для схемы построения криптографических хеш-функций SPONGE. Вестник КазНИТУ. – 2020. - N 5 (141). - С.645-650

[25] Оспанов Р.М. Киберщит: О внутренней функции в схеме построения криптографических хеш-функций «Sponge». Материалы Международной научно-практической Web-конференции «Военно-техническое обеспечение деятельности вооруженных сил: мировой опыт и тенденции развития», 20-22 июля 2020 года. - Нур-Султан: Из-во НУО, 2020. - С.351-353

[26] Berger T. P., Minier M., Pousse B. Software Oriented Stream Ciphers Based upon FCSRs in Diversified Mode. In: Roy B., Sendrier N. (eds) Progress in Cryptology - INDOCRYPT 2009. INDOCRYPT 2009. LNCS, vol.5922, Springer, Berlin, Heidelberg, 2009. - pp.119-135.

**REFERENCES**

[1] Mendel F., Pramstaller N., Rechberger C. A (second) preimage attack on the GOST hash function. In: Nyberg K. (eds) Fast Software Encryption. FSE 2008. LNCS, vol. 5086, Springer, Berlin, Heidelberg, 2008. - pp. 224–234.

[2] Mendel F., Pramstaller N., Rechberger C., Kontak M., Szmidt J. Cryptanalysis of the GOST hash function. In: Wagner D. (eds) Advances in Cryptology – CRYPTO 2008. CRYPTO 2008. LNCS, vol. 5157, Springer, Berlin, Heidelberg, 2008. - pp. 162–178.

[3] Wang X., Yao A.C., Yao F. Cryptanalysis on SHA-1. In: Proc. of NIST Cryptographic Hash Workshop (2005). October 31 - November 1, 2005. http://csrc.nist.gov/groups/ST/hash/documents/Wang_SHA1-New-Result.pdf.

[4] Wang X., Yin Y.L. Yu H. Finding collisions in the full SHA-1. In: Shoup V. (eds) Advances in Cryptology – CRYPTO 2005. CRYPTO 2005. LNCS, vol. 3621, Springer, Berlin, Heidelberg, 2005. - pp. 17-36.

[5] Stevens M., Karpman P., Peyrin T. Freestart collision for full SHA-1. IACR Cryptology ePrint Archive, Report 2015/967, 2015. https://eprint.iacr.org/2015/967.

[6] Stevens M., Bursztein E., Karpman P., Albertini A., Markov Y. The First Collision for Full SHA-1. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, LNCS, vol. 10401, Springer, Cham, 2017. - pp.570–596.

[7] Stevens M. Attacks on Hash Functions and Applications. PhD thesis, Mathematical Institute, Faculty of Science, Leiden University, 2012.

[8] Leurent G., Peyrin T. From Collisions to Chosen-Prefix Collisions Application to Full SHA-1. In: Ishai Y., Rijmen V. (eds) Advances in Cryptology – EUROCRYPT 2019. LNCS, vol. 11478, Springer, Cham, 2019. - pp. 527–555.

[9] Leurent G., Peyrin T. SHA-1 is a Shambles First Chosen-Prefix Collision on SHA-1 and Application to the PGP Web of Trust. IACR Cryptology ePrint Archive, Report 2020/014, 2020. https://eprint.iacr.org/2020/014.

[10] Albertini A., Aumasson J.-P., Eichlseder M., Mendel F., Schläffer M. Malicious hashing: Eve's variant of SHA-1. In: Joux, A., Youssef, A. (eds.) Selected Areas in Cryptography -- SAC 2014. SAC 2014. LNCS, vol. 8781, Springer, Cham, 2014. - pp. 1–19.

[11] Aumasson J.-P. Eve's SHA3 candidate: malicious hashing. (2011), https://www.aumasson.jp/data/papers/Aum11a.pdf.

[12] AlTawy R., Youssef A.M. Watch your constants: malicious streebog. IET Information Security, 9(6), 2015. - pp. 328–333.

[13] Morawiecki P. Malicious Keccak. IACR Cryptology ePrint Archive, Report 2015/1085, 2015. https://eprint.iacr.org/2015/1085.

[14] Shumow D., Ferguson N. On the possibility of a back door in the NIST SP800-90 Dual Ec Prng. In: CRYPTO 2007 Rump Session, 2007. http://rump2007.cr.yp.to/15- shumow.pdf.

[15] Barker E., Kelsey J. Recommendation for random number generation using deterministic random bit generators (revised). NIST Special Publication 800-90, 2007.

[16] Perrin L. Partitions in the S-Box of Streebog and Kuznyechik. IACR Transactions on Symmetric Cryptology, 2019(1). - pp. 302–329.

[17] National standard of the Russian Federation GOST R 34.12-2015 “Information Technology. Cryptographic data security. Block ciphers”, English version, Standartinform, Moscow, 2015.

[18] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: Ecrypt Hash Workshop 2007 (May 2007); available as public comment to NIST, http://www.csrc.nist.gov/pki/HashWorkshop/Public_Comments/2007_May.html

[19] Bertoni G., Daemen J, Peeters M., Van Assche G. Cryptographic sponge functions. Version 0.1, January 14, 2011. https://keccak.team/files/CSF-0.1.pdf.

[20] Bertoni G., Daemen J, Peeters M., Van Assche G. The Keccak reference. SHA-3 competition (round 3), 2011. https://keccak.team/sponge_duplex.html.

[21] Ospanov R.M., Seitkulov E.N., Arapov N.K., Ergalieva B.B. Modifikatsiya skhemy postroeniya kriptograficheskikh khesh-funktsii SPONGE. Vestnik KazNITU. - 2020. - N 5 (141). - C.520-525.

[22] Ospanov R.M., Seitkulov E.N. Kibershchit: O razlichnykh realizatsiyakh skhemy postroeniya kriptograficheskikh khesh-funktsii «Sponge». Materialy Mezhdunarodnoi nauchno-prakticheskoi Web-konferentsii «Voenno-tekhnicheskoe obespechenie deyatel'nosti vooruzhennykh sil: mirovoi opyt i tendentsii razvitiya», 20-22 iyulya 2020 goda. - Nur-Sultan: Iz-vo NUO, 2020. - S.305-308

[23] Wu H. The Hash Function JH. 2011. https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_round3.pdf.

[24] Ospanov R.M., Seitkulov E.N. O sposobakh proektirovaniya vnutrennei funktsii dlya skhemy postroeniya kriptograficheskikh khesh-funktsii SPONGE. Vestnik KazNITU. – 2020. - N 5 (141). - C.645-650

[25] Ospanov R.M. Kibershchit: O vnutrennei funktsii v skheme postroeniya kriptograficheskikh khesh-funktsii «Sponge». Materialy Mezhdunarodnoi nauchno-prakticheskoi Web-konferentsii «Voenno-tekhnicheskoe obespechenie deyatel'nosti vooruzhennykh sil: mirovoi opyt i tendentsii razvitiya», 20-22 iyulya 2020 goda. - Nur-Sultan: Iz-vo NUO, 2020. - S.351-353

[26] Berger T. P., Minier M., Pousse B. Software Oriented Stream Ciphers Based upon FCSRs in Diversified Mode. In: Roy B., Sendrier N. (eds) Progress in Cryptology - INDOCRYPT 2009. INDOCRYPT 2009. LNCS, vol.5922, Springer, Berlin, Heidelberg, 2009. - pp.119-135.

**Е.Н. Сейтқұлов*, Р.М. Оспанов, Б.Б.Ерғалиева**
Л.Н. Гумилев атындағы Еуразия ұлттық университеті, Нұр-Сұлтан, Қазақстан
*e-mail: yerzhan.seitkulov@gmail.com

## ӨЗГЕРТІЛГЕН SPONGE СХЕМАСЫНА НЕГІЗДЕЛГЕН КРИПТОГРАФИЯЛЫҚ ХЕШ ФУНКЦИЯСЫНЫҢ МЫСАЛЫ

**Андатпа.** Қазіргі уақытта «Sponge» схемасы заманауи криптографиялық хеш функцияларын құрудың ең сәтті және перспективалы тәсілі болып табылады. Бұл мақаланың мақсаты - осы схемаға негізделген криптографиялық хэш функциясының мысалын құру.. «Sponge» схемасының негізгі және маңызды құрамдас бөлігі-бұл бекітілген ұзындықты түрлендіру немесе функцияның ішкі күйін құрайтын биттердің белгіленген санымен жұмыс істейтін ішкі функция. Классикалық «Sponge» схемасы және оның көптеген модификациялары тек бір ішкі функцияны ұсынады. Бұл жұмыста көптеген ішкі функцияларды қолдануды қамтитын осы схемасының модификациясы қарастырылған. Ішкі функцияның үш жаңа нұсқасы қарастырылады. Біріншіден, айналмалы конфигурацияны беру үшін кері байланысы бар ығысу сөздік регистрлерін қолдануға негізделген ішкі функцияның нұсқасы қарастырылады. Екіншіден, Keccak ауыстыруына негізделген ішкі функцияның жаңа нұсқасы қарастырылады. Үшіншіден, AES жобалаудың жалпыланған әдіснамасы арқылы құрылған ішкі функция қарастырылады. Әрі қарай, модификацияланған схема негізінде және осы үш ішкі функцияны қолдана отырып, xtш алгоритмі құрылады. Бүкіл схеманың бөлігі ретінде осы үш ішкі функцияның біреуін таңдау жалған кездейсоқ түрде жасалған хабарламаға тәуелді таңдау биттерін қолдану арқылы анықталады.

**Негізгі сөздер:** ақпараттық қауіпсіздік, криптография, хеш функциясы, «Sponge» схемасы, ішкі функциясы.

**Y.N. Seitkulov*, R.M. Ospanov, B.B. Yergaliyeva**
Gumilyov Eurasian National University, Nur-Sultan, Kazakhstan
*e-mail: yerzhan.seitkulov@gmail.com

# AN EXAMPLE OF A CRYPTOGRAPHIC HASH FUNCTION BASED ON A MODIFIED SPONGE SCHEME

**Abstract.** Currently, the «Sponge» scheme is the most successful and promising way to build modern cryptographic hash functions. The purpose of this article is to build an example of a cryptographic hash function based on this scheme. The main and important component of the scheme is the internal function, which is a fixed-length transformation or permutation that operates with a fixed number of bits that make up the internal state of the function. The classic «Sponge» scheme and most of its modifications assume only one internal function. In this paper, we consider a modification of this scheme, which involves the use of a set of internal functions. Three new variants of the internal function are considered. First, we consider a variant of the internal function based on the use of word ring feedback with carry shift registers. Second, we consider a new version of the internal function based on the Keccak permutation. Third, we consider an internal function constructed using the generalized AES design methodology. Then, based on the modified scheme and using these three internal functions, a hash algorithm is constructed. The selection of one of these three internal functions as part of the entire scheme is determined using message dependent selection bits generated in a pseudo-random manner.

**Keywords:** information security, cryptography, hash function, «Sponge» scheme, internal function.