# ***GAGE and InGAGE***

v1.0

Submission to the NIST's Lightweight Cryptography Standardization Process

Designers/Submitters:

**Danilo Gligoroski**[1] - Department of Information Security and Communication Technology - IIK, Norwegian University of Science and Technology - NTNU, Trondheim, Norway,
Visiting: Department of Mathematics and Statistics, University of South Florida, USA

**Hristina Mihajloska**[2] - FCSE, "Ss Cyril and Methodius" University, Skopje, Republic of Macedonia

**Daniel Otte**[3] - Ruhr University, Bochum, Germany

`contact@gage.org`

26.03.2019

[1]address: O.S. Bragstads plass 2a 7491, Trondheim, Norway, tel: +47 73 59 43 24, email: danilog@ntnu.no

[2]address: "Rugjer Boshkovikj", 16, P.O. Box 393, 1000 Skopje, Republic of Macedonia, tel: +389 2 3070-377, email: hristina.mihajloska@finki.ukim.mk

[3]address: Amtsstr. 19, 44809 Bochum, Germany; tel: 0049 234 7948133; email: bg@nerilex.org;

# Table of Contents

# Chapter 1

# Hash Function Algorithm - GAGE

## 1.1 Specification

GAGE is a family of sponge-based hash functions [2] with states between 232 and 576 bits and rates for injecting message blocks of 8, 16, 32 and 64 bits. The sponge permutation has an SPN structure with one very light 4-to-2 bits s-box and a cheap hardware wiring i.e., bit-shuffling layer.

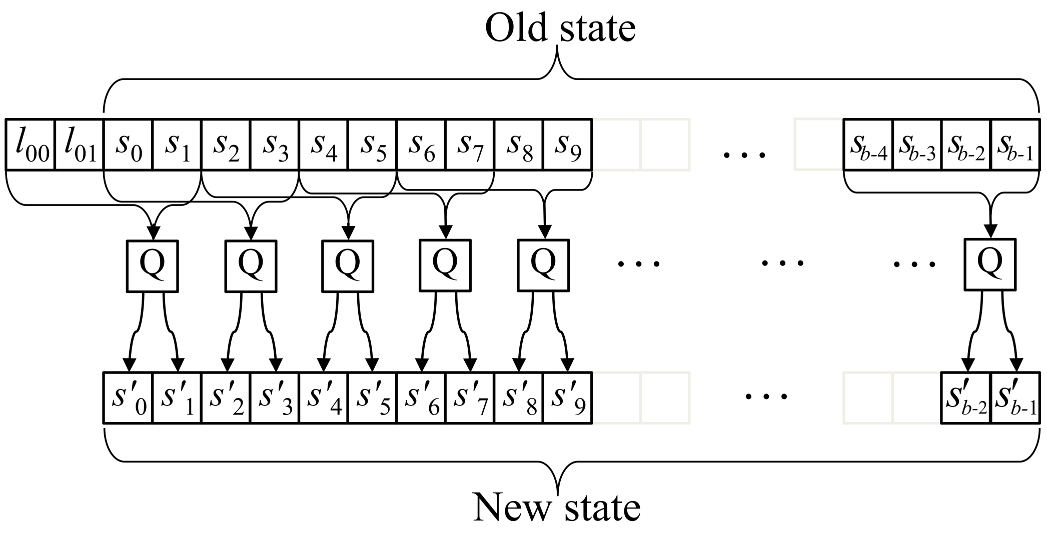
### 1.1.1 Round permutation

The round permutation consists of two parts: a nonlinear substitution part and a bit-shuffling part.

#### Nonlinear substitution part

The nonlinear substitution part uses one 4-to-2 bits s-box $Q$ that is applied in an interleaved way on the state of $b$ bits. Interleaved application means that the set of state bits is split in 2-bit subsets, and they enter the s-boxes in two different roles: as two left most bits and as two rightmost bits. The s-box is applied in parallel. This interleaved application makes the substitution layer as one big s-box with $(b + 2)$-to-$b$ bits. For transforming the two left most bits of the state, a two bit round constant $l_1 = (l_{0,1}, l_{1,1})$ is used. A graphical presentation of the substitution layer is given in Figure 1.1.

The 4-to-2 bits s-box $Q$ can be represented in a usual manner as one table with $2^4$ elements, where the elements are from the set $\{0, 1, 2, 3\}$ (given in Table 1.1).



Old state
$l_{00}$ $l_{01}$ $s_0$ $s_1$ $s_2$ $s_3$ $s_4$ $s_5$ $s_6$ $s_7$ $s_8$ $s_9$ … $s_{b-4}$ $s_{b-3}$ $s_{b-2}$ $s_{b-1}$
Q Q Q Q Q … … … Q
$s'_0$ $s'_1$ $s'_2$ $s'_3$ $s'_4$ $s'_5$ $s'_6$ $s'_7$ $s'_8$ $s'_9$ … $s'_{b-2}$ $s'_{b-1}$
New state

Figure 1.1: A graphical presentation of the nonlinear part in GAGE.

Table 1.1: The 4-to-2 bits s-box $Q$.

| 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 |

It can be also represented as in expression (1.1) as a $4 \times 4$ multiplication table for a binary operation $*$ over the set $Q = \{0, 1, 2, 3\}$ (Note: here we abuse and overload the use of the symbol $Q$ both as a symbol for the s-box and for a set.) More details about the algebraic structure $(Q, *)$ and two mutually inverse bijective transformations called $e$-transformation and $d$-transformation are given in the design rationale Section 1.5.

$$\begin{array}{c|cccc} * & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 0 & 3 & 2 \\ 1 & 0 & 2 & 1 & 3 \\ 2 & 2 & 3 & 0 & 1 \\ 3 & 3 & 1 & 2 & 0 \end{array} \tag{1.1}$$

Finally, the s-box $Q$ can be represented in ANF form as a vector-valued Boolean function that receives 4 input variables $x_1, x_2, x_3, x_4$ and outputs two Boolean functions: $Q(x_1, x_2, x_3, x_3) = (f_1(x_1, x_2, x_3, x_3), f_2(x_1, x_2, x_3, x_3))$ given with the expression (1.2).

Operations of addition and multiplication are in the finite filed $GF(2)$.

$$Q(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_2x_3 + x_2x_4,\ 1 + x_1 + x_2 + x_2x_3 + x_4 + x_2x_4) \tag{1.2}$$

We describe the nonlinear substitution part using the algebraic structure $(Q, *)$. For this, a state of $b$ bits that is subject of transformation is represented as an array $A = \{a_0, \ldots, a_{b/2-1}\}$ of two bit elements. The notation $x * y = z$ means $Q(x, y) = z$.

**Algorithm 1** Nonlinear substitution of $A = \{a_0, \ldots, a_{b/2-1}\}$ with $(Q, *)$ and a leader $l$.

```
1: procedure D-TRANSFORMATION($l$, $A = \{a_0, \ldots, a_{b/2-1}\}$ )
2:     $ldr \leftarrow l$
3:     for $i = 0$ to $b/2 - 1$ do
4:         $nextldr \leftarrow a_i$
5:         $a_i \leftarrow ldr * nextldr$
6:         $ldr \leftarrow nextldr$
7:     endfor
```

### Bit-shuffling part

The linear layer is just a procedure that shuffles the bits of the state. Let us represent a state of $b$ bits $S = (s_0, s_1, \ldots, s_{b-1})$ as an array of $b/8$ bytes $A = \{a_0, \ldots, a_{b/8-1}\}$, where $a_0 = (s_0, \ldots, s_7) \equiv [s_j, j = 0, \ldots, 7]$, $a_1 = (s_8, \ldots, s_{15}) \equiv [s_j, j = 8, \ldots, 15]$, …, $a_{b/8-1} = (s_{b-8}, \ldots, s_{b-1}) \equiv [s_j, j = b-8, \ldots, b-1]$. Let us also denote by $a_i[j]$ the $j$-th bit of the byte $a_i$ where the counting starts from $j = 0$ from the leftmost bit in $a_i$. Then every byte for the new state $A' = \{a'_0, \ldots, a'_{b/8-1}\}$ is described as follows:

$$a'_i = \pi_8([a_{(i+j)\,\mathbf{mod}\ b/8}[7-j], j = 0, \ldots 7]), \quad i = 0, \ldots, b/8 - 1 \tag{1.3}$$

The function $\pi_8()$ is the following permutation of 8 elements:

$$\pi_8 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 5 & 8 & 7 & 6 & 2 & 4 \end{pmatrix} \tag{1.4}$$

The bit-shuffling part is graphically presented in Figure 1.2.

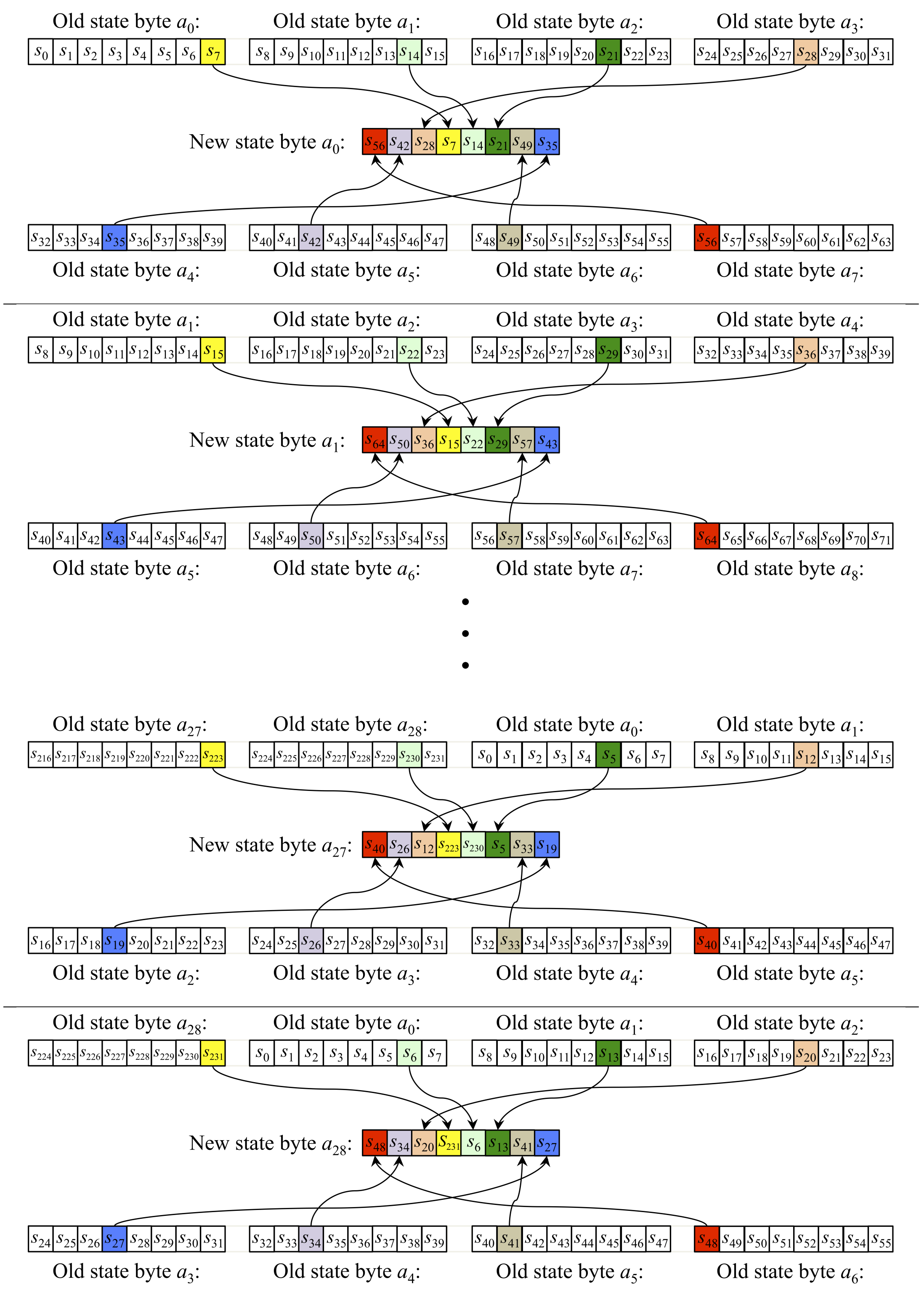


Figure 1.2: Bit-shuffling in GAGE, for a state of 232 bits i.e. 29 bytes.

### 1.1.2 SPN permutation

The complete permutation is a composition of alternating application of the substitution part and the bit-shuffling part for a number of rounds `ROUNDS`. For calling the substitution part we need to define an array of constants $Leaders = \{l_0, \dots, l_{\texttt{ROUNDS}-1}\}$. They are given in Table 1.2.

**Algorithm 2** SPN permutation of $A = \{a_0, \dots, a_{b/2-1}\}$ with `ROUNDS` rounds and with $Leaders = \{l_0, \dots, l_{\texttt{ROUNDS}-1}\}$.

```
1: procedure QPERMUTATION($A = \{a_0, \dots, a_{b/2-1}\}$, ROUNDS )
2:     D-TRANSFORMATION($l_0$, $A$)
3:     for $i = 1$ to ROUNDS do
4:         BIT-SHUFFLING($A$)
5:         D-TRANSFORMATION($l_i$, $A$)
6:     endfor
```

| $i$ | 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 |

Table 1.2: The list of 32 constant leaders. There is another longer list of 255 leaders given in Design Rationale.

### 1.1.3 Message padding

We give an algorithmic description of the message padding in Algorithm 3. Input parameters for the padding procedure are the message $M$, its length $mlen$ in bytes and the rate $r$. The output is a padded message $M_{pad}$ and the length of the padded message $\mu$ expressed as a number of blocks of length $r/8$ bytes.

**Algorithm 3** Message padding

```
1: procedure PADD($M = \{m_0, \dots, m_{len-1}\}, mlen, r$)
2:     $M_{pad} = M || \texttt{0x80} || (\texttt{0x00})^{((mlen+1) \mod \frac{r}{8})}$
3:     $\mu = \lceil \frac{mlen+1}{r/8} \rceil$
```

### 1.1.4 GAGE sponge

Table 1.3 gives all necessary information for building GAGE as a typical sponge-based hash function.

| Component | Notation | Relations/functions | Comment |
|---|---|---|---|
| Message | $M$ | $\|M\| = 8 \times mlen$ | $M$ is long $8 \times mlen$ bits i.e. $mlen$ bytes, where $mlen$ is always an integer. |
| State | $S$ | $\|S\| = b$, <br> $S = S_r \|\| S_c$ | State is split in two parts: rate part $S_r$ and capacity part $S_c$; $b$ is always a multiple of 8. |
| Rate | $r$ | $b = r + c$ | $r$ is always multiple of 8. |
| Capacity | $c$ | $b = r + c$ | |
| Padded message | $M_{pad}$ | $M_{pad} = M\|\|\texttt{0x80}\|\|(\texttt{0x00})^{(mlen+1) \mod \frac{r}{8}}$, <br> $M_{pad} \equiv M_1 \ \|\| \ \dots \ \|\| \ M_\mu$ | $M_{pad}$ is split in $\mu$ blocks, where every block is long $r/8$ bytes and $\mu = \lceil \frac{mlen+1}{r/8} \rceil$. |
| Initialization | | $S \leftarrow 0$ | All bits in the state are set to 0. |
| Absorbing | | 1: **for** $i = 1$ to $\mu$ **do** <br> 2: $\quad S \leftarrow (S_r \ \texttt{XOR} \ M_i) \ \|\| \ S_c$ <br> 3: $\quad S \leftarrow$ QPERMUTATION($S$, ROUNDS) <br> 4: **endfor** | |
| Squeezing | $Hash$ | 1: $H \leftarrow \varepsilon$ <br> 2: **for** $i = 1$ to $\frac{\|Hash\|}{r}$ **do** <br> 3: $\quad S \leftarrow$ QPERMUTATION($S$, ROUNDS) <br> 4: $\quad H \leftarrow H \ \|\| \ S[0, \dots, r-1]$ <br> 5: **endfor** <br> 6: $Hash \leftarrow H$ | The output is the string $Hash$. Symbol $\varepsilon$ denotes the empty string. |

Table 1.3: Components in GAGE sponge construction and their relations. Here, `0x80` denotes the byte with the value 128 i.e. a byte where all its bits are set to 0, except the leftmost i.e. the most significant bit which is set to 1, and `0x00` denotes the byte with the value 0.

## 1.2 Security Goals

Table 1.4 gives all instances of GAGE with security strengths for finding preimages, second preimages and collisions. The values for the security strengths are generic ones given in [1].

The highlighted row in Table 1.4 is **our main proposal** for the lightweight hash function: GAGE256 with a hash value of $n = 256$ bits, with an internal state of $b = 232$ bits, capacity of $c = 224$ bits and a rate $r = 8$ bits. It has a preimage security of 223 bits, a second preimage security of 112 bits and a collision security of 112 bits.

The message size limit in our proposal is set to $2^{64}$ bytes. It is the same length of the `unsigned long long` parameter set in the API interface. The input size (as a power of 2 bytes) can theoretically go up to the collision resistance value.

| Nr. | $\|Hash\| = n$ | $b$ | $c$ | $r$ | Preimage $\min(n, c-1)$ | 2nd preimage $\min(n, \frac{c}{2})$ | Collision $\frac{\min(n,c)}{2}$ | Message size limit (power of 2 bytes) |
|---|---|---|---|---|---|---|---|---|
| 1. | 256 | 232 | 224 | 8 | 223 | 112 | 112 | 64 |
| 2. | 256 | 240 | 224 | 16 | 223 | 112 | 112 | 64 |
| 3. | 256 | 256 | 224 | 32 | 223 | 112 | 112 | 64 |
| 4. | 256 | 288 | 224 | 64 | 223 | 112 | 112 | 64 |
| 5. | 256 | 272 | 256 | 16 | 256 | 128 | 128 | 64 |
| 6. | 256 | 288 | 256 | 32 | 256 | 128 | 128 | 64 |
| 7. | 256 | 320 | 256 | 64 | 256 | 128 | 128 | 64 |
| 8. | 256 | 384 | 256 | 128 | 256 | 128 | 128 | 64 |
| 9. | 256 | 544 | 512 | 32 | 256 | 256 | 128 | 64 |
| 10. | 256 | 576 | 512 | 64 | 256 | 256 | 128 | 64 |

Table 1.4: All instances of GAGE proposed in this documentation.

## 1.3 Security analysis

We first analyze the differential characteristics of the smallest nonlinear component in GAGE, its 4-to-2 s-box $Q$.

### 1.3.1 Differential characteristics of GAGE nonlinear layer

As an isolated component, the s-box $Q$ has very weak differential characteristics. It is given in Table 1.5. As we can see in cells highlighted in light red, there are 4 differentials in the table with probabilities of 100%. Let us represent the values of the input differentials (highlighted with light cyan color) for those 4 cells in binary with 4 bits: $\Delta_{max_1} = \{0, 0, 0, 0\}$, $\Delta_{max_2} = \{0, 0, 1, 1\}$, $\Delta_{max_3} = \{1, 0, 0, 0\}$ and $\Delta_{max_4} = \{1, 0, 1, 1\}$.

|  |  | $\Delta_{out}$ |  |  |  |
|---|---|---|---|---|---|
|  |  | 0 | 1 | 2 | 3 |
| $\Delta_{in}$ | 0 | 16 | 0 | 0 | 0 |
|  | 1 | 0 | 8 | 8 | 0 |
|  | 2 | 0 | 8 | 8 | 0 |
|  | 3 | 0 | 0 | 0 | 16 |
|  | 4 | 0 | 8 | 8 | 0 |
|  | 5 | 8 | 0 | 0 | 8 |
|  | 6 | 8 | 0 | 0 | 8 |
|  | 7 | 0 | 8 | 8 | 0 |
|  | 8 | 0 | 0 | 0 | 16 |
|  | 9 | 0 | 8 | 8 | 0 |
|  | 10 | 0 | 8 | 8 | 0 |
|  | 11 | 16 | 0 | 0 | 0 |
|  | 12 | 0 | 8 | 8 | 0 |
|  | 13 | 8 | 0 | 0 | 8 |
|  | 14 | 8 | 0 | 0 | 8 |
|  | 15 | 0 | 8 | 8 | 0 |

Table 1.5: Differential characteristics of the s-box $Q$.

In Algorithm 1 the s-box $Q$ in GAGE is used in an interleaved manner and it builds a huge s-box of size $(b + 2)$-to-$(b)$ bits. It is natural to check the differential characteristics of those s-boxes. In Table 1.6 we give a part of a differential distribution table for an 6-to-4 bits s-box obtained by the Algorithm 1 when the input length is 6 bits (2 bits for the leading constant, and 2 2-bit letters in the array $A = \{a_0, a_1\}$). If we represent again the values of the input differentials (highlighted with light cyan color) for those 4 cells in binary (now with 6 bits) we get: $\Delta_{max_1} = \{0, 0, 0, 0, 0, 0\}$, $\Delta_{max_2} = \{0, 0, 0, 0, 1, 1\}$, $\Delta_{max_3} = \{1, 0, 0, 0, 0, 0\}$ and $\Delta_{max_4} = \{1, 0, 0, 0, 1, 1\}$.

| | | Output | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 0 | 64 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 32 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 32 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0 | 0 | 64 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 4 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |
| | 5 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| | 6 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| | 7 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |
| | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 0 | 0 | 32 | 0 | 0 | 0 | 0 |
| | ⋮ | | | | | . | . | . | . | . | . | . | | | | | |
| | 28 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |
| | 29 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| | 30 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| n | 31 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |
| | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 64 | 0 | 0 | 0 |
| | 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 | 32 | 0 |
| p | 34 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 | 32 | 0 |
| | 35 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 64 |
| | 36 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |
| u | 37 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| | 38 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| | 39 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |
| | ⋮ | | | | | . | . | . | . | . | . | . | | | | | |
| | 56 | 0 | 0 | 0 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 |
| | 57 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 |
| | 58 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 |
| | 59 | 32 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 0 | 0 |
| t | 60 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |
| | 61 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| | 62 | 0 | 0 | 0 | 0 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 0 | 0 | 0 | 0 |
| | 63 | 0 | 0 | 0 | 0 | 0 | 16 | 16 | 0 | 0 | 16 | 16 | 0 | 0 | 0 | 0 | 0 |

Table 1.6: Differential characteristics of an 6-to-4 bits s-box obtained by the *d*-transformation $Q$.

**Proposition 1.** *There are exactly four cells with probabilities of 100% in the differential distribution table for an $(b+2)$-to-$b$ bits s-box obtained by the Algorithm 1 when the input length is $(b+2)$ bits (2 bits for the leading constant, and $b/2$ 2-bit letters in the array $A = \{a_0, \ldots, a_{b/2-1}\}$). The values of the input differentials for those 4 cells represented as integers as well as in binary with $b+2$ bits are:* $\Delta_{max_1} = 0 = \{\underbrace{0, 0, \ldots, 0, 0}_{b+2}\}$, $\Delta_{max_2} = 3 = \{\underbrace{0, \ldots, 0}_{b}, 1, 1\}$, $\Delta_{max_3} = 2^{b+1} = \{1, \underbrace{0, \ldots, 0}_{b+1}\}$ *and* $\Delta_{max_4} = 2^{b+1}+3 = \{1, \underbrace{0, \ldots, 0}_{b-1}, 1, 1\}$. □

While the first differential $\Delta_{max_1} = 0$ is a trivial one and not interesting for further analysis, the other three differentials for the sponge design of GAGE are impossible. More concretely, for $\Delta_{max_2} = 3 = \{\underbrace{0, \ldots, 0}_{b}, 1, 1\}$ the nonzero bits are far away from the zone of rate bits that can be controlled by the attacker, and for $\Delta_{max_3} = 2^{b+1} = \{1, \underbrace{0, \ldots, 0}_{b+1}\}$ and $\Delta_{max_4} = 2^{b+1} + 3 = \{1, \underbrace{0, \ldots, 0}_{b-1}, 1, 1\}$ the list of 32 constant leaders given in Table 1.2 are such that the `XOR` difference between any consecutive round leaders is not $\{1, 0\}$. A generating procedure that generates a longer list of 255 leading constants is given in Appendix 1. We summarize this discussion with the following Proposition.

**Proposition 2.** *For the sponge design GAGE, with constant leaders given in Table 1.2, the differentials* $\Delta_{max_2} = 3 = \{\underbrace{0, \ldots, 0}_{b}, 1, 1\}$, $\Delta_{max_3} = 2^{b+1} = \{1, \underbrace{0, \ldots, 0}_{b+1}\}$ *and* $\Delta_{max_4} = 2^{b+1} + 3 = \{1, \underbrace{0, \ldots, 0}_{b-1}, 1, 1\}$ *are impossible differentials.* □

The simple and systematic design of big $(b+2)$-to-$b$ bits s-box obtained by the Algorithm 1 offers a possibility to completely describe the distribution and the number of cells in its corresponding differential distribution table, regardless of its exponential size.

**Lemma 1.** *For a state of $b$ bits, the Algorithm 1 produces a differential distribution table where the number $N_{p=2^{-i}}$ of input differentials that have an output differential with a probability of $p_{\Delta_{out}} = 2^{-i}, i = 0, 1, \ldots, \frac{b}{2}$ is given by the following expression:*

$$N_{p=2^{-i}} = 4 \times 3^i \times \binom{b/2}{i}, \textit{ for } i = 0, 1, \ldots, \frac{b}{2}. \tag{1.5}$$

□

It is clear that the first part of Proposition 1 about the number of cells with probabilities of 100% in the differential distribution table, follows from Lemma 1 for $i = 0$.

**Corollary 1.**

$$\sum_{i=0}^{b/2} N_{p=2^{-i}} = 2^{b+2}. \tag{1.6}$$

□

**Corollary 2.** *For the $b$ bits state in GAGE, the number of cells with probabilities of $2^{-1} = 0.5$ in its differential distribution table is*

$$N_{2^{-1}} = 6 \times b. \tag{1.7}$$

□

Having Lemma 1 and Corollaries 1 and 2 gives us idea to define a discrete probability mass function for randomly selected input differences $\Delta_{in}$.

**Definition 1.** *Let the number of state bits is $b$ bits, and let $T$ be a differential distribution table produced for the $(b+2)$-to-$b$ bits s-box obtained by the Algorithm 1. Let $\Delta_{in}$ is a random variable describing a randomly selected input difference. The probability mass function $f_{\Delta_{in}}(i) = Pr\{\Delta_{in}$ has probability $2^{-i}$ of $\Delta_{out}$ in $T\}$ is given by the following expression:*

$$f_{\Delta_{in}}(i) = f(i) = \frac{N_{p=2^{-i}}}{2^{b+2}}. \tag{1.8}$$

In order to track differentials over several consecutive rounds of GAGE, we define a right skewed distribution confidence intervals as follows

**Definition 2.** *Let $b$ and $r$ represent the number of state bits and rate in GAGE, and let $f_{\Delta_{in}}(i)$ is its corresponding probability mass function for its differential distribution table. A maximal right skewed distribution confidence interval $\mathcal{I}_{max} = \mathcal{I}_{1-2^{-2r}} = [j_{min}, \frac{b}{2}]$ is the interval such that*

$$\sum_{j=0}^{j_{min}-1} f(j) \;<\; \frac{1}{2^{2r}} \tag{1.9}$$

*and*

$$\sum_{j=0}^{j_{min}} f(j) \;\geqslant\; \frac{1}{2^{2r}}. \tag{1.10}$$

The goal of the attacker using differential cryptanalysis is to find input differentials with as big as possible probability in the differential distribution table. For the sponge designs, the attacker has control over the rate bits in the first round, but does not have control over the next consecutive rounds. In order to set an upper bound of the probability of a differential over several consecutive rounds we use *the following plausible assumption*:

**Assumption 1.** *For GAGE instances that have $b$ state bits and $r$ rate bits, the input differentials for all rounds except the first round fall in the maximal right skewed distribution confidence interval $\mathcal{I}_{max}$.*

**Lemma 2.** *Let $S$ be a state of $b$ bits, $r$ is the rate, $\mathcal{I}_{max} = [j_{min}, \frac{b}{2}]$ is its maximal right skewed distribution confidence interval and let $R$ is the number of rounds for calling the procedure* QPERMUTATION$(S, R)$ *of GAGE. Let $P_{\Delta_{max}}(R)$ denotes the upper bound of input traced differentials for $R$ rounds. Under Assumption 1 it holds that*

$$P_{\Delta_{max}}(R) = \max\{2^{-(R-1)\times j_{min}-1}, 2^{-(b+2)}\}. \tag{1.11}$$

□

| | | | Traced differentials for $R = 1, 2, 3$ | | |
|---|---|---|---|---|---|
| $b$ | $r$ | $j_{min}$ | $P_{\Delta_{max}}(1)$ | $P_{\Delta_{max}}(2)$ | $P_{\Delta_{max}}(3)$ |
| 232 | 8 | 67 | $2^{-1}$ | $2^{-68}$ | $2^{-135}$ |
| 240 | 16 | 59 | $2^{-1}$ | $2^{-60}$ | $2^{-119}$ |
| 256 | 32 | 48 | $2^{-1}$ | $2^{-49}$ | $2^{-97}$ |
| 288 | 64 | 34 | $2^{-1}$ | $2^{-35}$ | $2^{-69}$ |
| 272 | 16 | 69 | $2^{-1}$ | $2^{-70}$ | $2^{-139}$ |
| 288 | 32 | 57 | $2^{-1}$ | $2^{-58}$ | $2^{-115}$ |
| 320 | 64 | 42 | $2^{-1}$ | $2^{-43}$ | $2^{-85}$ |
| 384 | 128 | 23 | $2^{-1}$ | $2^{-24}$ | $2^{-47}$ |
| 544 | 32 | 135 | $2^{-1}$ | $2^{-136}$ | $2^{-271}$ |
| 576 | 64 | 112 | $2^{-1}$ | $2^{-113}$ | $2^{-225}$ |

Table 1.7: Upper bounds of traced differentials for the first 3 rounds of GAGE for all proposed instances in this documentation.

We summarize findings from this section in Table 1.7, where we trace the upper bounds of the input differential probabilities for the first 3 rounds of GAGE. Finally, for $b = 232$ and $r = 8$ in Figure 1.3 we plot the probability mass function $f_{\Delta_{in}}(i)$ and its maximal right skewed distribution confidence interval. Note that the values on vertical axis for $f_{\Delta_{in}}(i)$ are presented in $\log_2$ scale.

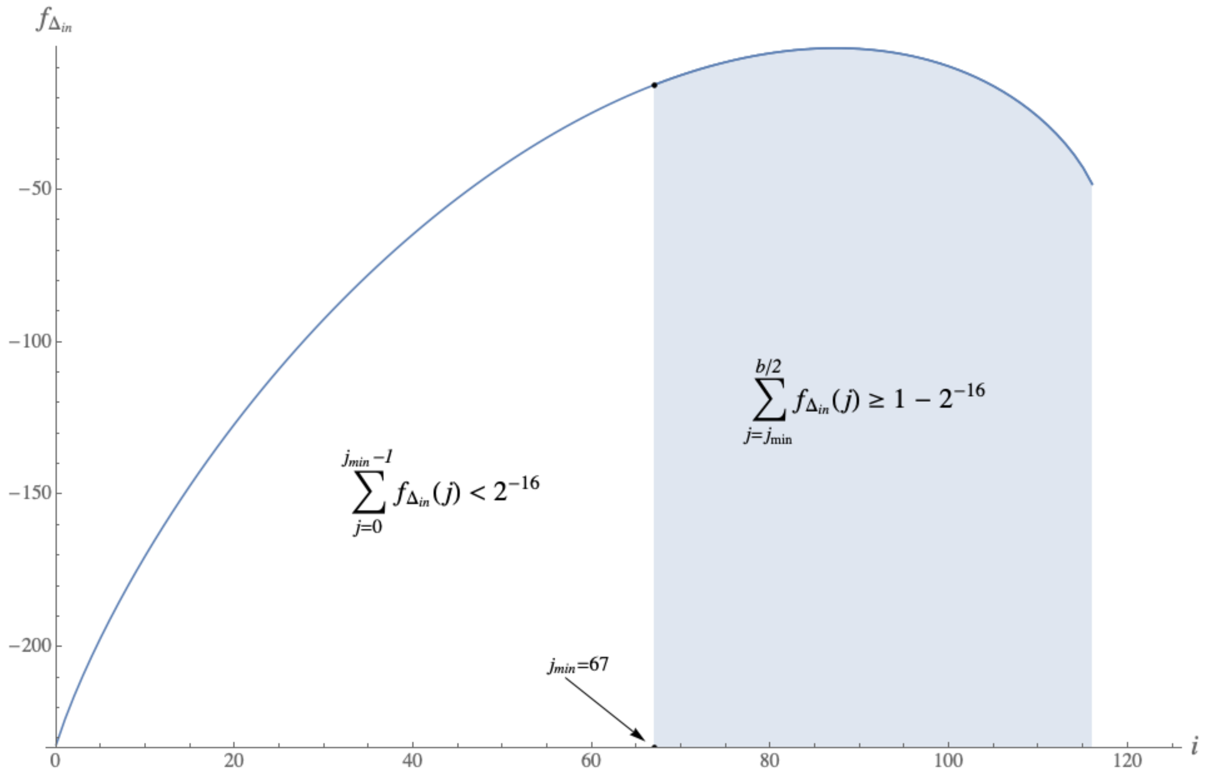


Figure 1.3: Log plot with base 2 of $f_{\Delta_{in}}(i)$ for $b = 232$ and $r = 8$.

### 1.3.2 Linear characteristics of GAGE

Similar to the differential characteristics, as an isolated component, the s-box $Q$ has very weak linear characteristics. It is given in Table 1.8. As we can see in cell highlighted in light red, there is one sum in the table in the row with index 7 which has the maximal extreme value -8.

| | | Output sum | | | |
|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 |
| | 0 | 8 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 0 | 0 |
| | 2 | 0 | 0 | 0 | 0 |
| | 3 | 0 | 0 | 0 | 0 |
| I | 4 | 0 | 0 | 0 | 0 |
| n | 5 | 0 | 0 | 0 | 0 |
| p | 6 | 0 | 0 | 0 | 0 |
| u | 7 | 0 | 0 | 0 | -8 |
| t | 8 | 0 | 0 | 0 | 0 |
| | 9 | 0 | -4 | 4 | 0 |
| s | 10 | 0 | 4 | 4 | 0 |
| u | 11 | 0 | 0 | 0 | 0 |
| m | 12 | 0 | 0 | 0 | 0 |
| | 13 | 0 | -4 | -4 | 0 |
| | 14 | 0 | -4 | 4 | 0 |
| | 15 | 0 | 0 | 0 | 0 |

Table 1.8: Linear approximation table for the s-box $Q$.

As we already mentioned in the previous section the Algorithm 1 uses the s-box $Q$ in GAGE in an interleaved manner and it builds a huge s-box of size $(b+2)$-to-$(b)$ bits. A direct production of a linear approximation table for that s-box is infeasible, but using the constant leaders defined in Table 1.2 it is possible to produce some linear approximation tables. Thus, in Table 1.9 we give a linear approximation table for an 4-to-4 s-box obtained from $Q$, the Algorithm 1 and the leader $l_0 = 0$. As we can see, there are seven extreme values (-8 and 8) highlighted in light red.

| | | Output sum | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -4 | 4 | 0 | 0 | 4 | 4 | 0 |
| | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 4 | 0 | 0 | 4 | -4 | 0 |
| | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 4 | 0 | 0 | 0 | 0 | -8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| n | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -4 | -4 | 0 | 0 | 4 | -4 | 0 |
| p | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -4 | 4 | 0 | 0 | -4 | -4 | 0 |
| u | 7 | 0 | 0 | 0 | -8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| t | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 9 | 0 | -4 | 4 | 0 | 0 | 4 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| s | 10 | 0 | 4 | 4 | 0 | 0 | 4 | -4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| u | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 |
| m | 12 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -8 | 0 | 0 | 0 |
| | 13 | 0 | -4 | -4 | 0 | 0 | 4 | -4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 14 | 0 | -4 | 4 | 0 | 0 | -4 | -4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -8 | 0 | 0 | 0 | 0 |

Table 1.9: Linear approximation table for an 4-to-4 s-box obtained from $Q$, the Algorithm 1 and leader $l_0 = 0$.

Having so many extreme values in these small linear approximation tables can give impression that there is some catastrophic weakness in the design of GAGE that is using the small s-box $Q$. However, these examples are not using the linear layer of GAGE. Since it is infeasible to work with the whole state of $b$ bits, and in order to analyze the linear characteristics of GAGE, we simulated a production of 8-to-8 s-boxes. For that purpose the state had $b = 8$ bits, we invoked the Algorithm 1 for different number of rounds using the leaders defined in Table 1.2 and we used one simple linear layer that rotates the bits in the state for one position to the right.

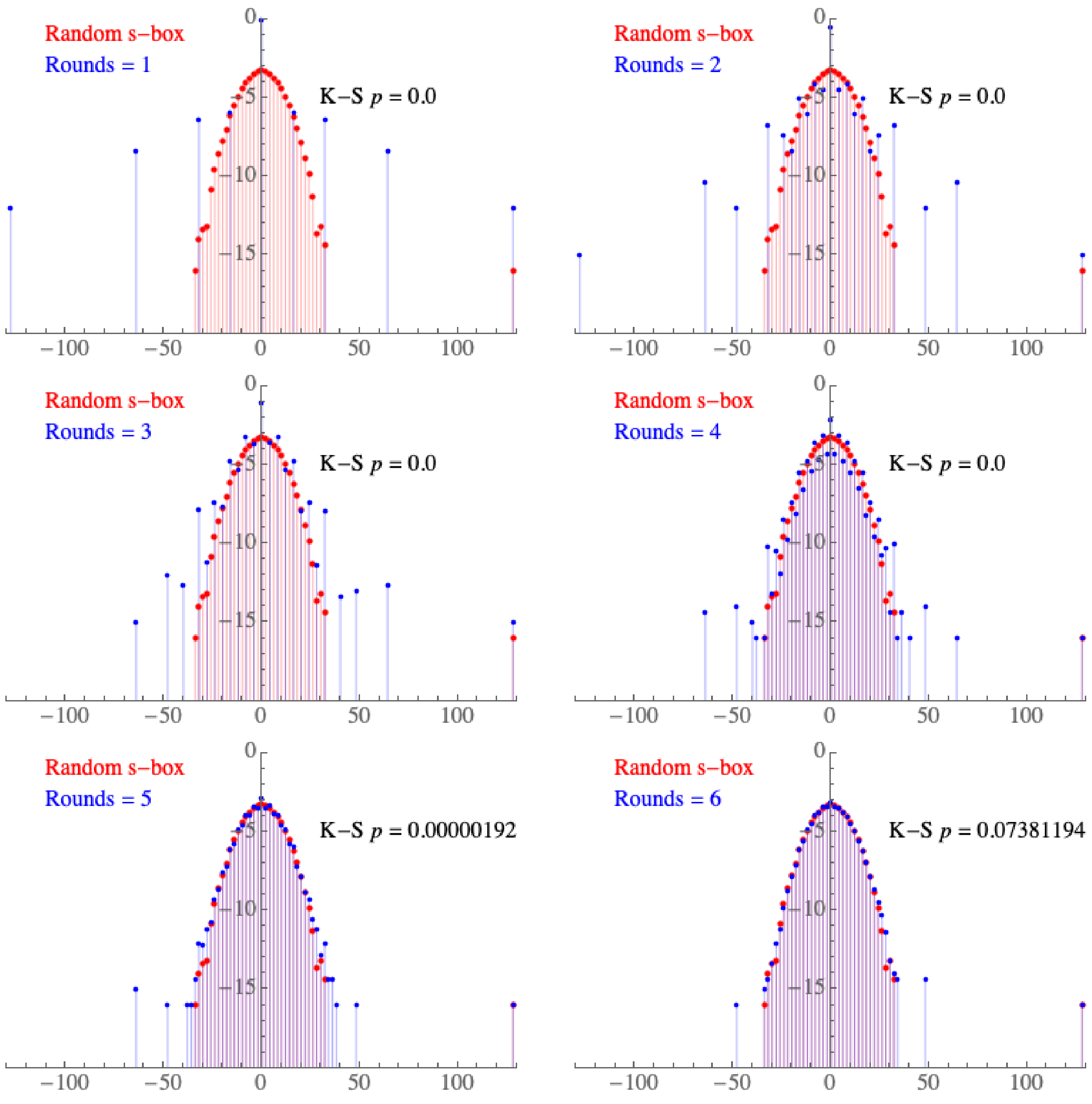


Figure 1.4: Comparative $Log_2$ plots of distribution of sums in a linear approximation tables for an 8-to-8 s-box obtained by Algorithm 1 with number of rounds from 1 to 8 (blue dots) and one randomly generated 8-to-8 s-box (red dots). The $p$ value is from comparing the distributions with two-sample Kolmogorov-Smirnov test.

We compared the linear characteristics of obtained simulated 8-to-8 s-boxes after different number of rounds to one linear approximation table for a randomly generated

8-to-8 s-box. The comparison is presented in Figures 1.4 and 1.5. We also show a $p$ value from a two-sample Kolmogorov-Smirnov test. Apparently, for small number of rounds, $p = 0.0$, but as we can see from the figures, the two distributions are getting close to each other as the number of rounds is increasing. For number of rounds more than 7 the statistical test gives a $p$ value which is not distinguishing the two samples.

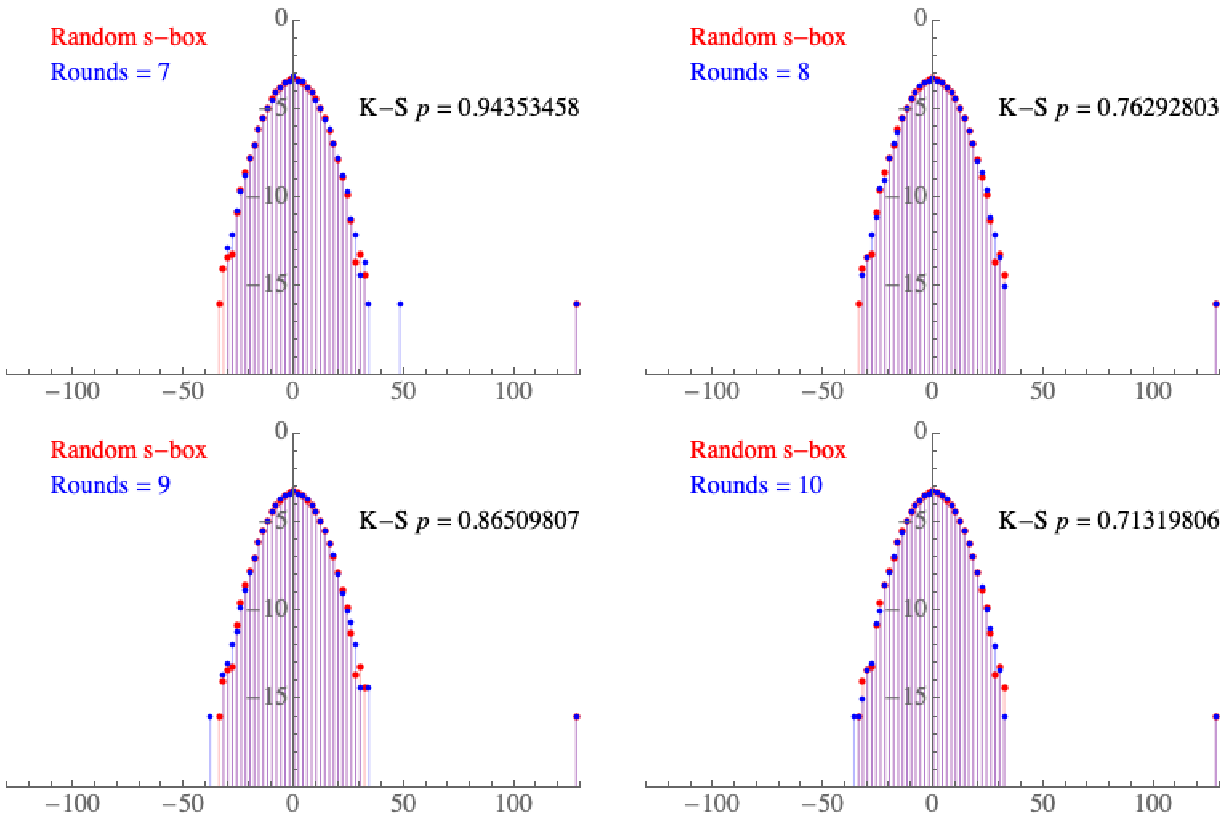


Figure 1.5: Comparative $Log_2$ plots with number of rounds from 7 to 10.

We want to stress that the tests we performed analyzing and comparing the linear characteristics of s-boxes coming from GAGE and a randomly generated s-box with the same size is not the ultimate proof of GAGE's resistance to linear cryptanalysis, but it certainly increases our confidence in the design, having in mind that the recommended number or rounds is `ROUNDS` $= 32$.

## 1.4 Features

1. GAGE is a sponge based cryptographic hash function with a state $S$ of $b$ bits, $b = r + c$ where $r$ is the rate and $c$ is capacity. $b$ is even number.

2. For the goal to be a lightweight crypto design, GAGE uses:
   - very small 4-to-2 bits s-box;
   - 2-bit round constants.

3. In most SPN crypto designs, substitution layers split the input bits in disjunctive subsets. Then those subsets are transformed in parallel by applying certain number of s-boxes. In GAGE, the set of input bits are split in 4-bit subsets, they are transformed in parallel by the s-box, **but those subsets are interleaved** in such a way that pairs of bits belong to two subsets. That makes the substitution layer to act as **one huge s-box** with an input of $b + 2$ bits, and $b$ bits of output.

4. The state $S$ in GAGE, can be represented either as:
   - a sequence of $\frac{b}{2}$, 2-bit cells (for very light hardware implementations);
   - a sequence of $\frac{b}{4}$, 4-bit cells (for unconventional 4-bit MCUs);
   - a sequence of $\frac{b}{8}$, 8-bit cells (for very small 8-bit MCUs);
   - a sequence of $\frac{b}{16}$, 16-bit cells (for small 16-bit MCUs);
   - a sequence of $\frac{b}{32}$, 32-bit cells (for 32-bit MCUs and CPUs);
   - a sequence of $\frac{b}{64}$, 64-bit cells (for 64-bit CPUs);
   - a $b$-bit sequence (for modern CPUs that have wide SIMD registers of 128, 256 or 512 bits).

   For all those representations with their different smallest basic cell (register), there exist operations over those cells that transform the state in equivalent manner. The nonlinear operations can be realized either as reading from small lookup tables or as register operations that perform only `SHIFT`, `XOR`, `AND` and `NOT` operations.

5. Its simple and systematic design offers a possibility to completely describe the distribution and the number of cells in the differential distribution table for the big $(b+2)$-to-$b$ bits s-box, regardless of the size of $b$.

6. GAGE can use two lines of developed protection techniques against side channel attacks:

    - If it is implemented using the small s-box $Q$ then the masking techniques for reading from look-up tables should be used [4]. The small size of the s-box enables to efficiently produce higher order masking look-up tables.
    - If it is implemented using the register operations of `SHIFT`, `XOR`, `AND` and `NOT`, then the Boolean masking techniques used for Keccak can be used also for GAGE.

## 1.5 Design rationale

We had several goals when designing GAGE: to design an ultra lightweight cryptographic hash function suitable for implementation in very low number of GEs in ASIC, in very low number of slices in FPGAs, in tiniest MCUs (even on 4-bit MCUs), but also to be competitively fast (if not faster) than other cryptographic hash functions on big CPUs. Since we wanted the design to share a lot of components with an AEAD cipher, the choice for the sponge design principles was natural.

The components in GAGE permutation, that by the way is built upon the SPN concept, are as simple and as with small hardware footprint as possible (if not the smallest possible):

1. the nonlinear substitution layer is using one 4-to-2 bits s-box in an interleaved mode of operation (named $d$-transformation) over the whole state.

2. the linear layer is just rewiring i.e. bit-shuffling. It is constructed in such a way that a full diffusion of an influence from one bit to all bits in the state is achieved in at least 9 rounds. It is also chosen to avoid early cyclic i.e. self-similarity repetition of the whole or any initial part of the sponge state that is longer than two bits.

3. the round constants are just 2-bit constants that form a non-repetitive sequence. In conjunction with the concept of a big $(b+2)$-to-$b$ bits s-box they are chosen to

prevent slide attacks, rotational or self-similarity attacks. They are also chosen to harden the design against differential cryptanalysis.

### 1.5.1 The nonlinear layer and the choice of the S-box

In the past we have designed the lightweight stream cipher Edon80 [7] that participated in Phase 3 of eSTREAM project. It uses four similar s-boxes, but the mode of operation in GAGE is different. We give here a brief overview of the mathematical properties for the chosen s-box in GAGE and the mode of operation for using that s-box.

**Definition 3.** *A quasigroup $(Q, *)$ is a groupoid satisfying the law*

$$(\forall u, v \in Q)(\exists!\ x, y \in Q) \quad u * x = v \ \&\ y * u = v. \tag{1.12}$$

What is characteristic for quasigroups is that equations of the type: $a * x = b$ or $x * a = b$ have unique solutions. The binary operation $*$ induces another binary operation on $Q$, called *left conjugate or left parastrophe*, defined as $x = a\backslash_* b$ iff $a * x = b$. It is obvious that $(Q, \backslash_*)$ is a quasigroup and that the algebra $(Q, *, \backslash_*)$ satisfies the identities

$$x\backslash_*(x * y) = y, \quad x * (x\backslash_* y) = y. \tag{1.13}$$

Consider an alphabet (i.e., a finite set) $Q$. By $Q^+$ we denote the set of all nonempty words (i.e., finite strings) formed by the elements of $Q$. Depending on the context, we use two notations for elements of $Q^+$: $a_1 a_2 \dots a_n$ and $(a_1, a_2, \dots, a_n)$, where $a_i \in Q$.

**Definition 4.** *Let $(Q, *)$ be a quasigroup and $M = a_1 a_2 \dots a_n \in Q^+$ For each $l \in Q$ we define two functions $e_{l,*}, d_{l,*} : Q^+ \longrightarrow Q^+$ as follows:*

$$e_{l,*}(M) = b_1 b_2 \dots b_n \Longleftrightarrow b_1 = l * a_1,\ b_2 = b_1 * a_2, \dots,\ b_n = b_{n-1} * a_n,$$

$$d_{l,*}(M) = c_1 c_2 \dots c_n \Longleftrightarrow c_1 = l * a_1,\ c_2 = a_1 * a_2, \dots,\ c_n = a_{n-1} * a_n,$$

*The functions $e_{l,*}$ and $d_{l,*}$ are called a quasigroup string $e$–transformation (or $e$–transformation for short) and a quasigroup string $d$–transformation (or $d$–transformation for short) of $Q^+$ based on the operation $*$ with leader $l$.*

Graphical representations of $e$–transformation and $d$–transformation are shown in Fig. 1.6.

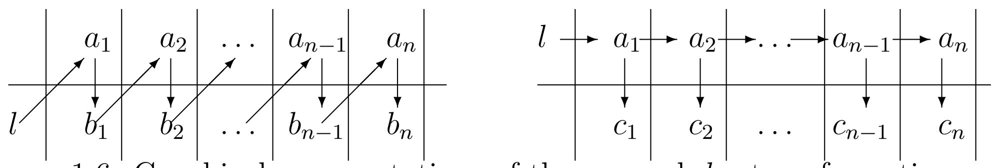


Figure 1.6: Graphical representations of the $e_{l,*}$ and $d_{l,*}$ transformations

Using Definition 4 and the identities (1.13) it is easy to prove the following theorem.

**Theorem 1.** *If $(Q, *)$ is a finite quasigroup, then $e_{l,*}$ and $d_{l,\backslash *}$ are mutually inverse permutations of $Q^+$, i.e.,*

$$d_{l,\backslash *}(e_{l,*}(M)) = M = e_{l,*}(d_{l,\backslash *}(M))$$

*for each leader $l \in Q$ and for every string $M \in Q^+$.* □

In [6] we proved that some of the well known modes of operations of block ciphers (such as CBC, OFB and CTR) are actually $e$–transformation or $d$–transformation. While the $e$–transformation is in essence a sequential procedure (as it is the case with the CBC encryption mode), the $d$–transformation is essentially a parallel procedure (a well known fact also for the decryption of the CBC mode). In contrast to the design of the stream cipher Edon80, the design of GAGE instead of the sequential $e$–transformation uses the parallel $d$–transformation.

Similar to Edon80, GAGE uses a small quasigroup of order 4 given in expression (1.1) or as a Boolean function in expression (1.2) and given here once more for the convenient reading.

| * | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | 0 | 3 | 2 |
| 1 | 0 | 2 | 1 | 3 |
| 2 | 2 | 3 | 0 | 1 |
| 3 | 3 | 1 | 2 | 0 |

$$Q(x_1, x_2, x_3, x_4) = (x_1 + x_3 + x_2x_3 + x_2x_4,\ 1 + x_1 + x_2 + x_2x_3 + x_4 + x_2x_4)$$

There are 576 quasigroups of order 4, and if we sort all of them in a lexicographic order, in GAGE we are using the quasigroup Nr. 173. Here are the criteria for choosing this particular quasigroup:

1. The quasigroup and its left conjugate (left parastrophe) should be nonlinear Boolean functions;

2. The quasigroup should not have fixed points;

3. The quasigroup should give as little as possible cells with 100% probability in its differential distribution table and in differential distribution tables obtained from its corresponding $d$–transformation.

### 1.5.2 Linear layer of GAGE permutation

Linear layers in SPN constructions can be as simple as just bit-shuffling, or can be complex linear operations among the bits of the state. The advantage of having complex linear operations in the linear layer is in faster diffusion of the influence that each bit of the state has. However, more complex linear layer needs bigger hardware footprint when implemented in hardware. If the goal is a lightweight hardware design, then the cheapest linear layer for that purpose is just a simple rewiring of the bits i.e. bit-shuffling. That is the design choice in GAGE.

Additionally, for the linear layer in GAGE we used the following four design criteria:

1. Bit-shuffling should be efficient also on small 8-bit MCUs; For this purpose every new byte in the state is composed of bits that come from 8 different bytes in the old state.

2. Bit-shuffling should be one and same procedure for different sizes of the state; No slight differences in constants (except the size of the state), no differences in rotations or other operations that perform the linear layer bit-shuffling;

3. Bit-shuffling should be such that there is no early cyclic repetition of the state after few rounds of the SPN design;

4. Bit-shuffling should be such that in combination with the nonlinear layer, the diffusion in the state is as fast as possible.

Having design criteria Nr. 1 we found the following bit-shuffling procedure (already given in (1.3) and (1.4)) and repeated here for convenient reading. Every byte for the new state $A' = \{a'_0, \ldots, a'_{b/8-1}\}$ is described as follows:

$$a'_i = \pi_8([a_{(i+j)\,\mathbf{mod}\ b/8}[7-j], j = 0, \ldots 7]), \quad i = 0, \ldots, b/8-1$$

The function $\pi_8()$ is the following permutation of 8 elements:

$$\pi_8 = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 3 & 5 & 8 & 7 & 6 & 2 & 4 \end{pmatrix}$$

**Definition 5.** *Let $S = (s_0, s_1, \ldots, s_{b-1})$ be a state of $b$ bits and let $\pi_S \equiv \pi$ is a permutation of $b$ elements. Let $Pairing(S) = S_{pairs} = \{\{s_0, s_1\}, \{s_2, s_3\}, \ldots, \{s_{b-2}, s_{b-1}\}\}$ be a procedure that splits the state $S$ in disjunctive union of neighboring two-elements subsets. We say that $\pi_S$ has a pair-matching period $f$ if there is a pair $\{s_{2\nu}, s_{2\nu+1}\}$ such that*

$$\{s_{2\nu}, s_{2\nu+1}\} \in Pairing(\underbrace{\pi(\pi(\ldots\pi(S)))}_{f}). \tag{1.14}$$

**Definition 6.** *For a state $S$ of $b$ bits, a maximal pair-matching permutation is a permutation $\pi$ on $b$ elements that has a pair-matching period $\frac{b}{2}$.*

About the design criteria Nr. 3 we searched for a bit-shuffling permutation $\pi_S$ such that it is a maximal pair-matching permutation.

About the design criteria Nr. 4, in Table 1.10 we give a list of number of rounds for every instance proposed in this documentation. Having in mind that $d$–transformation in Algorithm 1 combines always quartets of bits as input to the s-box $Q$ it is an expectable phenomena to need more rounds, as the size of the state increases, in order to achieve a full diffusion.

| $b$ | $c$ | $r$ | Nr of rounds for full diffusion |
|---|---|---|---|
| 232 | 224 | 8 | 9 |
| 240 | 224 | 16 | 9 |
| 256 | 224 | 32 | 10 |
| 288 | 224 | 64 | 10 |
| 272 | 256 | 16 | 10 |
| 288 | 256 | 32 | 10 |
| 320 | 256 | 64 | 11 |
| 384 | 256 | 128 | 12 |
| 544 | 512 | 32 | 16 |
| 576 | 512 | 64 | 16 |

Table 1.10: Number of rounds for different sizes of the state, for achieving a full diffusion

### 1.5.3 Round constants in GAGE

For generating the 2-bit round constants in GAGE we used an 8-bit LFSR defined by the irreducible polynomial $x^8 + x^6 + x^5 + x^4 + 1$. Generating a bit sequence from an LFSR is out of the scope of this document (and can be found in any basic textbook for LFSRs), but for our purposes we refer to two LFSR procedures:

1. Procedure InitLFSR(`LFSRState`) initializes the 8-bit register with some state `LFSRState`;
2. Procedure LFSR(`&LFSRState`) returns the current value of the least significant bit of the state, and then updates the state acording to the rules of the LFSR.

Using LFSR with 8-bit state can produce a non-repeating sequence of 255 bits. We used the output from the LFSR in Algorithm 4 to generate 255 2-bit constants from the set $\{0, 1, 2, 3\}$. While we need only 32 constants, we generated the full set of 255 values for any further use. The list of constants are generated in such a way that their values are chosen in an alternating way: If the previous constant was chosen from the set $\{0, 2\}$, then the current constant (indexed by an index $i$) will be chosen from the set $\{1, 3\}$ depending on the output bit from the LFSR.

**Algorithm 4** Generating constants in GAGE.

```
1:  LFSRState ← 0xAA
2:  InitLFSR(LFSRState)
3:  $i \leftarrow 0$
4:  $jump \leftarrow 0$
5:  do
6:      $bit \leftarrow$ LFSR(&LFSRState)
7:      if $jump == 0$ then
8:          if $bit == 0$ then
9:              leader$[i++] \leftarrow 0$
10:         else
11:             leader$[i++] \leftarrow 2$
            $jump \leftarrow 1$
12:     else
13:         if $bit == 0$ then
14:             leader$[i++] \leftarrow 1$
15:         else
16:             leader$[i++] \leftarrow 3$
            $jump \leftarrow 0$
17: while LFSRState $\neq$ 0xAA
```

Note that we start from an LFSR state that generates the first leader $l_0 = 0$. The reasons for choosing constants either from $\{0, 2\}$ or from $\{1, 3\}$ come from our analysis of differential distribution tables of s-boxes obtained by Algorithm 1.

First, let us give the following simple Corollary:

**Corollary 3.** *If* $x \in \{0, 2\}$ *and* $y \in \{1, 3\}$ *then* $(x \texttt{ XOR } y) \in \{1, 3\}$.

Next, recall four differentials with 100% probabilities: $\Delta_{max_1} = 0 = \{\underbrace{0, 0, \ldots, 0, 0}_{b+2}\}$, $\Delta_{max_2} = 3 = \{\underbrace{0, \ldots, 0}_{b}, 1, 1\}$, $\Delta_{max_3} = 2^{b+1} = \{1, \underbrace{0, \ldots, 0}_{b+1}\}$ and $\Delta_{max_4} = 2^{b+1} + 3 = \{1, \underbrace{0, \ldots, 0}_{b-1}, 1, 1\}$.

By starting with leader $l_0 = 0$ we make the differential $\Delta_{max_2} = \{\underbrace{0, \ldots, 0}_{b}, 1, 1\}$ impossible since the two nonzero bits are far away from the left-most zone of rate bits. In such case, also the differentials $\Delta_{max_3}$ and $\Delta_{max_4}$ are impossible. But after the first initial round where the attacker has some limited control over the input bits, in the subsequent rounds we would like the XOR difference in the first two bits to never be 0 or 2.

There comes the Corollary 3, and the reasons why we generate the leader constants in an alternating manner as described in Algorithm 4.

Finally in Table 1.11 we give the list of all 255 leader constants generated by Algorithm 4.

| $i$ | $l_i$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 3 | 0 | 3 | 0 | 1 | 2 | 3 | 2 | 1 | 2 | 3 | 2 | 1 | 2 | 3 |
| 16 | 0 | 1 | 2 | 3 | 2 | 3 | 0 | 3 | 2 | 3 | 2 | 3 | 2 | 1 | 2 | 1 |
| 32 | 0 | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 2 | 1 | 2 | 1 | 0 | 1 | 2 | 3 |
| 48 | 0 | 1 | 0 | 1 | 0 | 3 | 2 | 3 | 0 | 3 | 0 | 3 | 0 | 3 | 0 | 3 |
| 64 | 2 | 3 | 2 | 3 | 0 | 1 | 2 | 1 | 2 | 1 | 0 | 1 | 0 | 3 | 0 | 1 |
| 80 | 2 | 3 | 2 | 3 | 2 | 3 | 2 | 3 | 0 | 1 | 0 | 1 | 2 | 1 | 2 | 3 |
| 96 | 2 | 3 | 0 | 1 | 0 | 3 | 2 | 1 | 2 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 112 | 2 | 1 | 0 | 1 | 2 | 3 | 2 | 1 | 0 | 1 | 2 | 1 | 0 | 3 | 0 | 3 |
| 128 | 2 | 3 | 0 | 1 | 0 | 1 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 1 | 0 | 3 |
| 144 | 0 | 1 | 2 | 3 | 0 | 3 | 2 | 3 | 0 | 1 | 2 | 1 | 0 | 1 | 0 | 1 |
| 160 | 2 | 1 | 2 | 1 | 2 | 3 | 0 | 3 | 2 | 1 | 2 | 1 | 2 | 3 | 0 | 1 |
| 176 | 2 | 1 | 2 | 3 | 0 | 1 | 0 | 1 | 2 | 3 | 2 | 3 | 2 | 1 | 2 | 3 |
| 192 | 0 | 3 | 2 | 3 | 2 | 1 | 2 | 1 | 2 | 3 | 2 | 1 | 2 | 1 | 0 | 1 |
| 208 | 2 | 1 | 0 | 1 | 0 | 3 | 2 | 1 | 2 | 3 | 0 | 1 | 0 | 3 | 2 | 3 |
| 224 | 2 | 1 | 0 | 3 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 0 | 3 | 0 | 3 |
| 240 | 2 | 1 | 2 | 1 | 0 | 3 | 0 | 1 | 0 | 3 | 0 | 3 | 0 | 1 | 2 | |

Table 1.11: A list of 255 leaders produced by Algorithm 4.

# Chapter 2

# AEAD Algorithm - InGAGE

InGAGE is a sponge-based family of authenticated ciphers with associated data built over the lightweight cryptographic hash function GAGE. InGAGE mode of operation is similar to Ascon family of authenticated ciphers [5] (which is similar to MonkeyDuplex modes of operation [3]). Still, InGAGE has several differences with Ascon both in its mode of operation and in used parameters.

## 2.1 Specification

InGAGE uses the standard 4-parameter AEAD interface discussed in [11] and in [10]. Its encryption function is described as

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

where $K$ is a secret key, $N$ is a nonce, $A$ is an associated data, $P$ is a plaintext, $C$ is a ciphertext with same length as the plaintext and $T$ is an authentication tag. Its decryption function is described as

$$\mathcal{D}(K, N, A, C, T) = P \text{ or } \textsc{Invalid}$$

The modes of encryption and decryption of InGAGE are given in Figures 2.1 and 2.2. The invocation of the sponge permutation $\textsc{qpermutation}()$ in InGAGE is done with the following two values: $r_1 = \texttt{ROUNDS}$ and $r_2 = \texttt{ROUNDS}/2$ where the value `ROUNDS` is defined for the hash function GAGE (in this submission has value 32).

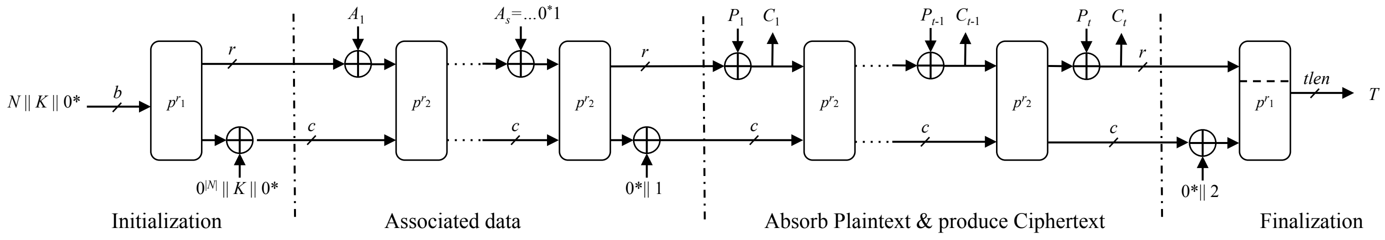


Figure 2.1: A graphical presentation of InGAGE encryption mode of operation.

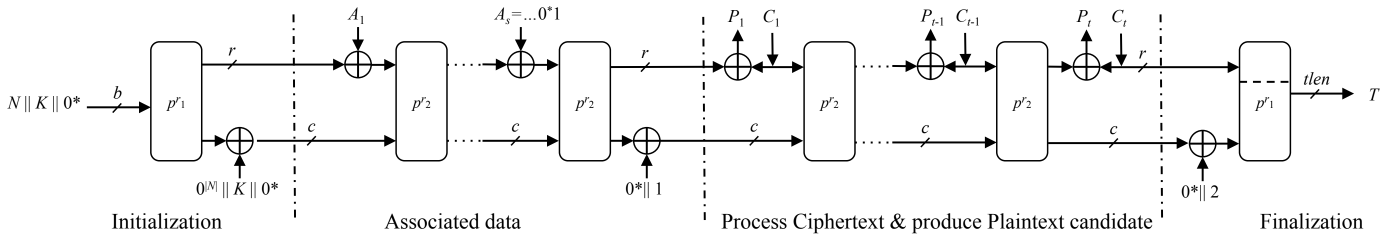


Figure 2.2: A graphical presentation of InGAGE decryption mode of operation.

In algorithmic form InGAGE encryption and decryption are given in Algorithm 5 and 6. Note that the names of the variables for some of the InGAGE parameters are common and shared with the hash function GAGE (such as the state $S$ which is split in the rate and in the capacity part i.e. $S = S_r || S_c$ or the parameter `ROUNDS`) while the other names are changed or new are added due to the nature of the encryption algorithm i.e. instead of an input message $M$ we have plaintext message $P$. Also we have a ciphertext message $C$, and the authentication tag $T$ has length of $tlen$ bits. The notation $S[r, \dots r + tlen - 1]$ means that we take the first $tlen$ bits of capacity part $S_c$ of the state $S$ (right after the first $r$ bits that belong to the rate part $S_r$) when it is represented as sequences of bits.

**Algorithm 5** InGAGE authenticated encryption $\mathcal{E}(K, N, A, P)$.

**Input:** secret key $K$, nonce $N$, associated data $A$, plaintext $P$

**Output:** ciphertext $C$, authentication tag $T$ of $tlen$ bits

```
Initialize state $S$
1: $S \leftarrow N||K||0^*$
2: $S \leftarrow \textsc{qpermutation}(S, \texttt{ROUNDS})$
3: $S \leftarrow 0||K||0^*$
Absorb associated data $A$
1: $\{A_{pad}, \mu\} \leftarrow \textsc{Padd}(A, alen, r)$
2: $A_\mu \leftarrow A_\mu$ XOR $(0^*||\texttt{0x01})$
3: for $i = 1$ to $\mu$ do
4:     $S \leftarrow (S_r$ XOR $A_i) \; || \; S_c$
5:     $S \leftarrow \textsc{qpermutation}(S, \texttt{ROUNDS/2})$
6: endfor
7: $S \leftarrow S$ XOR $(0^*||\texttt{0x01})$
Absorb and encrypt the plaintext $P$
1: $\{P_{pad}, \mu\} \leftarrow \textsc{Padd}(P, plen, r)$
2: for $i = 1$ to $\mu$ do
3:     $C_i \leftarrow (S_r$ XOR $P_i)$
4:     $S \leftarrow C_i \; || \; S_c$
5:     $S \leftarrow \textsc{qpermutation}(S, \texttt{ROUNDS/2})$
6: endfor
7: $S \leftarrow S$ XOR $(0^*||\texttt{0x02})$
Finalization
1: $S \leftarrow \textsc{qpermutation}(S, \texttt{ROUNDS})$
2: $T \leftarrow S[r, \ldots, r + tlen - 1]$
```

**Algorithm 6** InGAGE verified decryption $\mathcal{D}(K, N, A, C, T)$.

**Input:** secret key $K$, nonce $N$, associated data $A$, ciphertext $C$, tag $T$ of $tlen$ bits

**Output:** plaintext $P$ or INVALID

```
Initialize state $S$
1: $S \leftarrow N||K||0^*$
2: $S \leftarrow$ QPERMUTATION($S$, ROUNDS)
3: $S \leftarrow 0||K||0^*$
Absorb associated data $A$
1: $\{A_{pad}, \mu\} \leftarrow$ PADD($A, alen, r$)
2: $A_\mu \leftarrow A_\mu$ XOR ($0^*||$0x01)
3: for $i = 1$ to $\mu$ do
4:     $S \leftarrow (S_r$ XOR $A_i) \ || \ S_c$
5:     $S \leftarrow$ QPERMUTATION($S$, ROUNDS/2)
6: endfor
7: $S \leftarrow S$ XOR ($0^*||$0x01)
Process the ciphertext $C$
1: $\{C_{pad}, \mu\} \leftarrow$ PADD($C, clen, r$)
2: for $i = 1$ to $\mu$ do
3:     $P_i \leftarrow (S_r$ XOR $C_i)$
4:     $S \leftarrow C_i \ || \ S_c$
5:     $S \leftarrow$ QPERMUTATION($S$, ROUNDS/2)
6: endfor
7: $S \leftarrow S$ XOR ($0^*||$0x02)
Finalization
1: $S \leftarrow$ QPERMUTATION($S$, ROUNDS)
2: $T' \leftarrow S[r, \ldots, r + tlen - 1]$
3: if $T == T'$ then
4:     Return $P$
5: else
6:     Return INVALID
```

## 2.2 Security Goals

Table 2.1 gives all proposed instances of InGAGE with their security goals. The values for the security strengths are generic ones given in [1].

The highlighted row in Table 2.1 is **our main proposal** for the lightweight AEAD function: a key of $|K| = 128$ bits, a nonce of $|N| = 96$ bits, a tag length of $|T| = 128$, with an internal state of $b = 232$ bits, capacity of $c = 224$ bits and a rate $r = 8$ bits. It has a confidentiality of plaintext security of 128 bits, integrity of plaintext of 128 bits and integrity of associated data security of 128 bits. It is not designed to work with nonce reusing.

The message size limit in our proposal is set to $2^{64}$ bytes. It is the same length of the `unsigned long long` parameter set in the API interface. The input size (as a power of 2 bytes) can theoretically go up to the collision resistance value.

| Nr. | $\|K\|$ | $\|N\|$ | $\|T\|$ | $b$ | $c$ | $r$ | Confidentiality of $P$ | Integrity of $P$ | Integrity of $A$ | Nonce reuse | Message size limit (power of 2 bytes) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1. | 128 | 96 | 128 | 232 | 224 | 8 | 116 | 128 | 128 | No | 64 |
| 2. | 128 | 96 | 128 | 240 | 224 | 16 | 120 | 128 | 128 | No | 64 |
| 3. | 128 | 96 | 128 | 256 | 224 | 32 | 128 | 128 | 128 | No | 64 |
| 4. | 128 | 128 | 128 | 320 | 256 | 64 | 128 | 128 | 128 | No | 64 |
| 5. | 256 | 96 | 128 | 512 | 448 | 64 | 256 | 128 | 128 | No | 64 |
| 6. | 256 | 128 | 128 | 512 | 448 | 64 | 256 | 128 | 128 | No | 64 |

Table 2.1: All instances of InGAGE proposed in this documentation.

## 2.3 Security analysis

InGAGE security utterly rely on the security of the cryptographic function GAGE (discussed in the first part of this documentation), as well as the security of MonkeyDuplex modes of operation [3], Ascon mode of operation [5] and the analysis of sponge based AEAD that offers security beyond $2^{c/2}$ as described in [9].

### 2.3.1 Security loss of a reused nonce in InGAGE

Let $K$ be a secret key, $A$ be an associated data, and $N$ be a nonce that will be misused for two or more plaintexts. While the recovery of the secret key $K$ is not feasible due to the used MonkeyDuplex and à la Ascon modes of operation, there is a security loss of a reused nonce $N$ in InGAGE. That security loss is completely described in Rogaway et al., paper [8] with the trivial attack described in Section 1 of that paper and with the CPSS (chosen-prefix, secret-sufix) attack in Section 3 of that paper. In any case the security loss of missused nonces in InGAGE can be summarized as follows:

- (trivial attack in [8]) For InGAGE with a rate parameter $r$, having a ciphertext $C = C_1 C_2 \ldots C_t T$, and having an access to the encryption oracle $\mathcal{E}(K, N, A, P)$, it is possible to recover the corresponding plaintext $P$ with a complexity $O((2^r - 1)t)$ queries.
- (CPSS attack in [8]) For InGAGE with a rate parameter $r$, recovering messages of a format $M = P||S$, where $P$ part is a known prefix controlled by the adversary, and $S$ is not known by the adversary, has a complexity $O(2^r |S|/r)$ queries.
- It is not feasible to recover the key $K$ by reused nonce in InGAGE.

### 2.3.2 Summary of the relevant attacks on InGAGE

#### Impossible differentials and related key attacks

Differentials of the form $\Delta_{max_2} = 3 = \{\underbrace{0, \ldots, 0}_{b}, 1, 1\}$ have probability of 100%. Thus, the attacker that has control over the inputs of the last two bits of the state can use these differentials to launch a related key attack scenario. In order to render these differentials as impossible in InGAGE the initialization of the state is defined as: $S \leftarrow N||K||0^*$.

#### Traced differentials for all proposed instances of InGAGE

Based on the security analysis of traced differentials for the hash function GAGE, we give here a Table 2.2 of upper bounds of traced differentials for the first 3 rounds of all proposed instances of InGAGE.

| $b$ | $r$ | $j_{min}$ | Traced differentials for $R = 1, 2, 3$ $P_{\Delta_{max}}(1)$ | $P_{\Delta_{max}}(2)$ | $P_{\Delta_{max}}(3)$ |
|---|---|---|---|---|---|
| 232 | 8 | 67 | $2^{-1}$ | $2^{-68}$ | $2^{-135}$ |
| 240 | 16 | 59 | $2^{-1}$ | $2^{-60}$ | $2^{-119}$ |
| 256 | 32 | 48 | $2^{-1}$ | $2^{-49}$ | $2^{-97}$ |
| 320 | 64 | 42 | $2^{-1}$ | $2^{-43}$ | $2^{-85}$ |
| 512 | 64 | 94 | $2^{-1}$ | $2^{-95}$ | $2^{-189}$ |

Table 2.2: Upper bounds of traced differentials for the first 3 rounds of all proposed instances of InGAGE.

### Collision attacks on MonkeyDuplex round reduced permutation

Since InGAGE is a sponge based AEAD with MonkeyDuplex mode of operation [3], for the phases of associated data absorption and plaintext encryption it is using a round reduced permutation with `ROUNDS/2` rounds. One possible attack for this case is an attack that is producing collisions in the capacity part $S_c$ of the state. In that case, the attacker by controlling the rate part $S_r$ can produce forged authentication tags. To mitigate these attacks, the parameter `ROUNDS` is set to 32, and thus `ROUNDS/2` is 16. The full diffusion of the influences of every bit of the state to every other bit of the state is given in Table 2.3, and we see that 16 rounds are more than enough for all instances of InGAGE to achieve a full diffusion. Achieving a full diffusion after 16 rounds, in conjunction with the differential and linear characteristics of the permutation function GAGE, means that it is unlikely that there exist high probability differentials that will lead to collisions in $S_c$.

| $b$ | $c$ | $r$ | Nr of rounds for full diffusion |
|---|---|---|---|
| 232 | 224 | 8 | 9 |
| 240 | 224 | 16 | 9 |
| 256 | 224 | 32 | 10 |
| 320 | 256 | 64 | 11 |
| 512 | 448 | 64 | 15 |

Table 2.3: Number of rounds for different sizes of the state, for achieving a full diffusion for all instances of InGAGE

**Reused nonce in InGAGE**

In case of reused nonce, InGAGE losses completely its security and an adversary can recover the plaintext with a complexity $O(2^r)$ queries where $r$ is the rate of the sponge function used in InGAGE.

## 2.4 Features

1. InGAGE is a lightweight sponge based AEAD with a state $S$ of $b$ bits, $b = r + c$ where $r$ is the rate and $c$ is capacity ($b$ is even number), nonces of 96 or 128 bits, and secret keys of 128 or 256 bits.

2. For the goal to be a lightweight crypto design, InGAGE uses the lightweight sponge based hash function GAGE.

3. InGAGE is nonce respecting cipher but a nonce misuse will not result in a catastrophic key recovery. This is due to its use of MonkeyDuplex and à la Ascon modes of operation.

4. The state $S$ in InGAGE, can be represented in different ways (from 2-bit cells up to long SIMD registers) simmilar as GAGE.

5. For all those representations with their different smallest basic cell (register), there exist operations over those cells that transform the state in equivalent manner.

The nonlinear operations can be realized either as reading from small tables or as register operations that perform only `SHIFT`, `XOR`, `AND` and `NOT` operations.

6. InGAGE can use two lines of developed protection techniques against side channel attacks:

    - If it is implemented using the small s-box $Q$ then the masking techniques for reading from look-up tables should be used [4]. The small size of the s-box enables to efficiently produce higher order masking look-up tables.
    - If it is implemented using the register operations of `SHIFT`, `XOR`, `AND` and `NOT`, then the Boolean masking techniques used for Keccak can be used also for InGAGE.

## 2.5 Design rationale

We decided to design a lightweight AEAD standing on the shoulders of the theoretical and practical developments in the field of cryptographic sponge designs [1, 2, 3] in the last decade. That is why we first designed a very light cryptographic hash function GAGE and then we applied its permutation function in a similar manner as the elegant design of Ascon and by considering the excellent theoretical results for the security of sponge based AEAD beyond $2^{c/2}$, presented in [9].

The mode of InGAGE differs from Ascon mode in the following parts:

- It has state that has different sizes from 232 bits up to 512 bits, and it has different rates (from 8 bits up to 64 bits).
- The initialization of the state is in the form $S \leftarrow N||K||0^*$. We especially wanted to avoid injecting the key material at the rightmost bits of the state due to our differential cryptanalysis of GAGE. If we would allow the key to be injected to the rightmost part, then there would be one possibility of launching a key-related attack to produce the differential $\Delta_{max_2} = 3 = \{\underbrace{0, \ldots, 0}_{b}, 1, 1\}$.
- Padding the associated data and padding the plaintext is different in InGAGE. More concretely, after calling the padding procedure $\textsc{Padd}()$, the rightmost byte of the padded associated data is changed to end with the bit 1 (instead of the situation for the plaintext where the padding end with the bit 0).

- Absorption phases for the associated data and for the plaintext end with different flipped bits in the state: For the associated data it is the rightmost bit of the state that is flipped i.e. $S \leftarrow S$ `XOR` $(0^*||$`0x01`$)$, but for the plaintext it is the second to the rightmost bit of the state that is flipped i.e. $S \leftarrow S$ `XOR` $(0^*||$`0x02`$)$.

- The finalization part without injecting the key $K$ invokes the sponge permutation with a full number of `ROUNDS`. Our rationale is that since the number of `ROUNDS` is used for the cryptographic hash function GAGE, there is no extra security gain if we inject the key material before invoking the sponge permutation.

- Tag is produced by extracting the leftmost bits of the capacity part of the state $S_c$.

# Chapter 3

# Reference Implementation for GAGE and InGAGE

The reference implementations of GAGE and InGAGE are written in C programming language (C99 version). They can be accessed via standard git interface from the following URL link: `https://c0de.pw/lwc-mipri/reference-c/tree/master/`.

Here we basically give the same information as it is given in the README.md file on the git repository.

## 3.1 Building/compiling the code

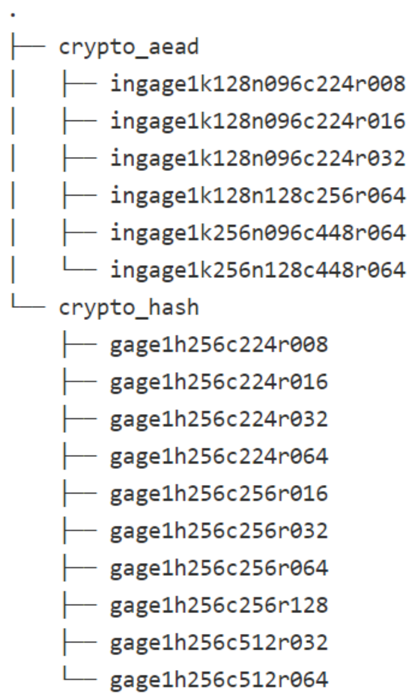
The package comes with a `Makefile` which is intended to be used with the `make` tool under Linux or similar operating systems.

You can use `make TARGET` to build `TARGET`. The following targets are defined:

- `all`: build the reference implementations.
- `kat`: generate KAT (Known Answer Test) vectors.
- `check`: check generated KAT vectors against supplied checksums.
- `UPDATE_REFERENCE_KAT_VECTORS`: update the reference checksums, this should be only used if the specification changes.
- `release`: create zip files according to NIST requirements.

## 3.2 Directory structure

The release directory structure is shown on Figure 3.1.



```
.
├── crypto_aead
│   ├── ingage1k128n096c224r008
│   ├── ingage1k128n096c224r016
│   ├── ingage1k128n096c224r032
│   ├── ingage1k128n128c256r064
│   ├── ingage1k256n096c448r064
│   └── ingage1k256n128c448r064
└── crypto_hash
    ├── gage1h256c224r008
    ├── gage1h256c224r016
    ├── gage1h256c224r032
    ├── gage1h256c224r064
    ├── gage1h256c256r016
    ├── gage1h256c256r032
    ├── gage1h256c256r064
    ├── gage1h256c256r128
    ├── gage1h256c512r032
    └── gage1h256c512r064
```

Figure 3.1: Release directory structure

The directories for the Gage and InGage reference implementations are similarly structured. They are structured in the following way:

1. First level distinguishes between hash functions in `crypto_hash` and AEAD functions in `crypto_aead`.
2. Second level distinguishes the different members of each family.
3. Third level directories offer different implementation variants with `ref` being the reference implementation.

Each directory name on the second level starts with the families name (gage or ingage) followed by a version number and several parameters. Each parameter is indicated by a single letter followed by a numeric value.

Parameters for GAGE:

- h⟨num⟩: ⟨num⟩ is the number of bits of the resulting message digest
- c⟨num⟩: ⟨num⟩ is the value of the capacity parameter
- r⟨num⟩: ⟨num⟩ is the value of the rate parameter

Parameters for InGAGE:

- k⟨num⟩: ⟨num⟩ is the size of the key in bits
- n⟨num⟩: ⟨num⟩ is the size of the nonce in bits
- c⟨num⟩: ⟨num⟩ is the value of the capacity parameter
- r⟨num⟩: ⟨num⟩ is the value of the rate parameter

The structure of the reference implementation is shown on Figure 3.2.

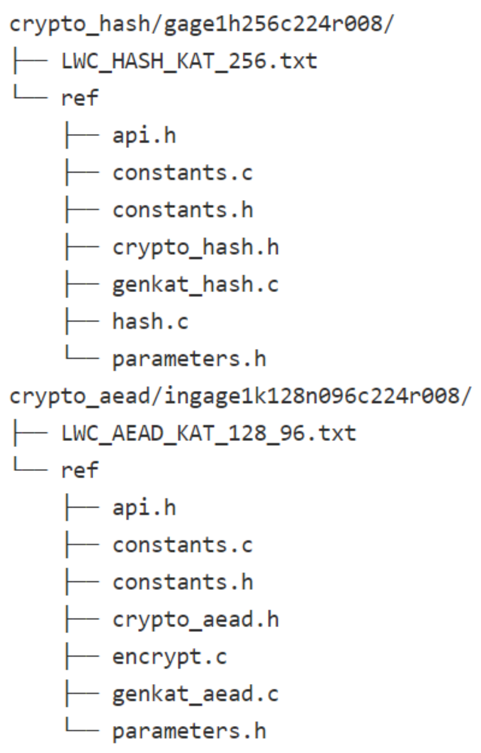
## 3.3 Paired Hash-AEAD proposals and shared programming code

In Table 3.1 we give a list of 4 paired Hash-AEAD proposals that share the same values for $b$, $c$ and $r$.

| GAGE Nr. | $\|Hash\| = n$ | $b$ | $c$ | $r$ | $\|K\|$ | $\|N\|$ | $\|T\|$ | InGAGE Nr. |
|---|---|---|---|---|---|---|---|---|
| 1. | 256 | 232 | 224 | 8 | 128 | 96 | 128 | 1. |
| 2. | 256 | 240 | 224 | 16 | 128 | 96 | 128 | 2. |
| 3. | 256 | 256 | 224 | 32 | 128 | 96 | 128 | 3. |
| 7. | 256 | 320 | 256 | 64 | 128 | 128 | 128 | 4. |

Table 3.1: All instances of paired GAGE - InGAGE proposals.

In addition, we would like to emphasize that reference implementations of GAGE and InGAGE family members share a large amount of code. The members of GAGE



```
crypto_hash/gage1h256c224r008/
├── LWC_HASH_KAT_256.txt
└── ref
    ├── api.h
    ├── constants.c
    ├── constants.h
    ├── crypto_hash.h
    ├── genkat_hash.c
    ├── hash.c
    └── parameters.h
crypto_aead/ingage1k128n096c224r008/
├── LWC_AEAD_KAT_128_96.txt
└── ref
    ├── api.h
    ├── constants.c
    ├── constants.h
    ├── crypto_aead.h
    ├── encrypt.c
    ├── genkat_aead.c
    └── parameters.h
```

Figure 3.2: Structure of the reference implementation

and InGAGE families differ only in their state size and how this size is split in rate and capacity. Those values are defined for each implementation in `parameters.h`.

Since InGAGE also offers four different interfaces ($k = 128, n = 96; k = 128, n = 128; k = 256, n = 96; k = 256, n = 128$), there are also four different variations of `api.h` which define those parameters.

All implementations share:

- `constants.c`: this file defines the leaders or round constants and the s-box.
- `constants.h`: provides interfaces to the constants defined in `constants.c` and defines the state structure.

All implementations of GAGE further share:

- `hash.c`: the core implementation of GAGE.
- `api.h`: defines the size of the message digest to 32 bytes.

- `crypto_hash.h`: provides a prototype for the *crypto_hash*() function.
- `genkat_hash.c`: provides a main program to generate test vectors.

All implementations of InGAGE share:

- `encrypt.c`: the core implementation of InGAGE.
- `crypto_aead.h`: provides a prototype for the *crypto_aead_encrypt*() and *crypto_aead_decrypt*() functions.
- `genkat_aead.c`: provides a main program to generate test vectors.

# Chapter 4

# Statements

## 4.1 Statement by Each Submitter

I, *Danilo Gligoroski*, of *Department of Information Security and Communication Technology - IIK, Norwegian University of Science and Technology - NTNU, Trondheim, Norway, currently visiting: Department of Mathematics and Statistics, University of South Florida, USA*, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as GAGE/InGAGE, is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystems specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment I do hereby agree to provide the statements required by Sections 2.4.2 and 2.4.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the lightweight crypto evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.4.1, 2.4.2 and 2.4.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Danilo Gligoroski
Title: Professor
Date: 25 February 2019
Place: Tampa, Florida, USA

I, *Hristina Mihajloska*, of *FCSE, "Ss Cyril and Methodius" University, Skopje, Republic of Macedonia*, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as GAGE/InGAGE, is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystems specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment I do hereby agree to provide the statements required by Sections 2.4.2 and 2.4.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the lightweight crypto evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.4.1, 2.4.2 and 2.4.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Hristina Mihajloska
Title: Assistant Professor
Date: 25 February 2019
Place: Skopje, Macedonia

I, *Daniel Otte*, of *Ruhr University, Bochum, Germany*, do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as GAGE/InGAGE, is my own original work, or if submitted jointly with others, is the original work of the joint submitters.

I further declare that I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations.

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystems specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment I do hereby agree to provide the statements required by Sections 2.4.2 and 2.4.3, below, for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the lightweight crypto evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.4.1, 2.4.2 and 2.4.3, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Daniel Otte
Title: Master student
Date: 25 February 2019
Place: Bochum, Germany

## 4.2 Statement by Patent (and Patent Application) Owner(s)

**N/A**

## 4.3 Statement by Reference/Optimized/Additional Implementations Owner(s)

I, *Danilo Gligoroski*, of *Department of Information Security and Communication Technology - IIK, Norwegian University of Science and Technology - NTNU, Trondheim, Norway, currently visiting: Department of Mathematics and Statistics, University of South Florida, USA*, am the owner of the submitted reference implementation, optimized and additional implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the lightweight cryptography public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed: Danilo Gligoroski
Title: Professor
Date: 25 February 2019
Place: Tampa, Florida, USA

I, *Hristina Mihajloska*, of *FCSE, "Ss Cyril and Methodius" University, Skopje, Republic of Macedonia*, am the owner of the submitted reference implementation, optimized and additional implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the lightweight cryptography public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed: Hristina Mihajloska
Title: Assistant Professor
Date: 25 February 2019
Place: Skopje, Macedonia

I, *Daniel Otte*, of *Ruhr University, Bochum, Germany*, am the owner of the submitted reference implementation, optimized and additional implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the lightweight cryptography public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed: Daniel Otte
Title: Master student
Date: 25 February 2019
Place: Bochum, Germany

# Acknowledgments

The first author would like to thank Prof. Natasha Jonoska and the Department of Mathematics and Statistics, University of South Florida, USA, for their hospitality. This document was produced during his sabbatical visit of that institution.

The authors want to thank NIST people for their useful comments on the pre-submission documentation of GAGE and InGAGE from 04 January 2019.

# References

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, 2007.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak sponge function family main document. *Submission to NIST (Round 2)*, 3(30), 2009.

[3] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Permutation-based encryption, authentication and authenticated encryption. *Directions in Authenticated Ciphers*, 2012.

[4] Jean-Sébastien Coron. Higher order masking of look-up tables. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 441–458. Springer, 2014.

[5] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1. 2. *Submission to the CAESAR Competition*, 2016.

[6] Danilo Gligoroski, Suzana Andova, and Svein Johan Knapskog. On the importance of the key separation principle for different modes of operation. In *International Conference on Information Security Practice and Experience*, pages 404–418. Springer, 2008.

[7] Danilo Gligoroski, Smile Markovski, and Svein Johan Knapskog. The stream cipher Edon80. In *New Stream Cipher Designs*, pages 152–169. Springer, 2008.

[8] Viet Tung Hoang, Reza Reyhanitabar, Phillip Rogaway, and Damian Vizr. Online authenticated-encryption and its nonce-reuse misuse-resistance. Cryptology ePrint Archive, Report 2015/189, 2015. `https://eprint.iacr.org/2015/189`.

[9] Philipp Jovanovic, Atul Luykx, and Bart Mennink. Beyond $2^{c/2}$ security in sponge-based authenticated encryption modes. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 85–104. Springer, 2014.

[10] David McGrew. An interface and algorithms for authenticated encryption. Technical report, IETF, 2008.

[11] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 98–107. ACM, 2002.