

Lightweight Linear Layers with High Branch Number for Ascon-Like Permutations

DOI: <https://doi.org/10.54654/isj.v1i27.6404>

Hoang Dinh Linh, Tran Thi Luong*, Nguyen Van Long

Abstract— The linear layer is a fundamental building block of substitution-permutation network (SPN) based ciphers, responsible for the diffusion of differences and linear masks across the state. In this paper, we analyze and evaluate several lightweight linear layers through the lens of branch number theory, focusing on constructions that rely solely on XOR operations and cyclic rotations, such as the Column Parity Mixer (CPM), the Twin Column Parity Mixer (TCPM), the Symmetric Twin Column Parity Mixer (STCPM), and the Multiple Rows Mixer (MRM). Based on these analyses, we propose new linear layers that achieve higher branch numbers while requiring fewer XOR operations than existing designs. We then evaluate the proposed constructions when instantiated in Ascon-like permutations operating on a 5×64 -bit state, comparing Ascon, Gaston, Gaston-S, and Hsilu in terms of trail weight bounds. Our results confirm that the proposed linear layers achieve: a column differential branch number 7 at 2.8 XOR/bit (Construction 1), branch number 7 with improved linear resistance at 3 XOR/bit (Construction 2), and branch number 17 at 4 XOR/bit (Construction 3). In particular, the final structure provides a substantially higher column branch number compared to TCPM and STCPM at a similar XOR implementation cost.

Tóm tắt— Lớp tuyến tính là một thành phần cơ bản trong các hệ mật dựa trên mạng thay thế – hoán vị (Substitution-Permutation Network - SPN), chịu trách nhiệm khuếch tán các sai khác và mặt nạ tuyến tính trên toàn bộ trạng thái. Trong bài báo này, chúng tôi phân tích và đánh giá một số lớp tuyến tính nhẹ dưới góc độ lý thuyết số nhánh, tập trung vào các cấu trúc chỉ sử dụng các phép XOR và dịch vòng, như Column Parity Mixer (CPM),

Twin Column Parity Mixer (TCPM), Symmetric Twin Column Parity Mixer (STCPM) và Multiple Rows Mixer (MRM). Dựa trên các phân tích này, nhóm tác giả đề xuất các lớp tuyến tính mới đạt được số nhánh cao hơn trong khi yêu cầu ít phép XOR hơn so với các thiết kế hiện có. Tiếp theo, chúng tôi đánh giá các cấu trúc đề xuất khi được cài đặt trong các hoán vị kiểu Ascon hoạt động trên trạng thái kích thước 5×64 -bit, so sánh với Ascon, Gaston, Gaston-S và Hsilu về các cận trọng số đường đặc trưng. Kết quả cho thấy các lớp tuyến tính đề xuất đạt được: số nhánh vi sai theo cột bằng 7 với chi phí 2.8 phép XOR/bit (Cấu trúc 1); số nhánh bằng 7 với khả năng kháng tuyến tính được cải thiện ở mức 3 XOR/bit (Cấu trúc 2); và số nhánh bằng 17 tại 4 XOR/bit (Cấu trúc 3). Đặc biệt, cấu trúc cuối đạt số nhánh cột cao hơn đáng kể so với các cấu trúc có chi phí XOR tương đương như TCPM và STCPM.

Keyword—Lightweight cryptography; linear layer; branch number; XOR and rotation; permutation; Ascon; column parity mixer; multiple rows mixer.

Từ khóa—Mật mã hạng nhẹ; tầng tuyến tính; số nhánh; XOR và dịch vòng; hoán vị; Ascon; bộ xáo trộn chặn lẻ cột; bộ xáo trộn đa hàng.

I. INTRODUCTION

Designing secure and efficient cryptographic primitives for resource-constrained environments is one of the most practically relevant challenges in symmetric cryptography today. Devices such as RFID tags, IoT sensors, and embedded microcontrollers impose strict limits on power consumption, gate area, and computational throughput, while resisting differential and linear cryptanalysis. These competing demands have motivated a long line of research into lightweight cryptography, culminating in the NIST standardization of Ascon in 2023 [1].

Symmetric ciphers following the SPN paradigm alternate a non-linear layer (providing confusion through S-boxes) with a linear layer (providing diffusion). Besides their classical

This manuscript was received on April 29, 2026. It was reviewed June 4, 2026, revised on June 17, 2026 and accepted on June 22, 2026.

* Corresponding author

cryptanalytic properties, S-boxes have recently been investigated from the perspective of quantum implementation complexity and ancilla-free circuit constructions [2, 3]. The linear layer is frequently the dominant factor in the implementation cost of lightweight primitives, and its security contribution is captured by the branch number: a small branch number admits low-weight differential or linear trails that survive many rounds, enabling statistical attacks. Designing a linear layer that simultaneously achieves a high branch number and a low XOR count is therefore a central problem in lightweight cipher design.

A. Design Strategies for Linear Layers

There are two main strategies for building a cryptographic linear layer. The first relies on Maximum Distance Separable (MDS) matrices over extension fields, which attain the theoretically optimal branch number for their dimension but incur non-trivial field multiplication costs. For example, a 4×4 MDS matrix over $\text{GF}(2^d)$ achieves branch number 5 at approximately $2 + 3/(4d)$ XOR/bit [4], and practical dimensions are limited to 8×8 [5]. The second strategy uses only XOR and cyclic bit rotations. This approach underlies many prominent designs including Keccak [6], Xoodoo [7], and Ascon [8], and offers very low cost at the expense of a smaller branch number.

B. Related Work

Ascon's linear layer σ operates independently on each row. Specifically, every row A_i is updated by XOR-ing it with two rotated copies of itself, where the rotation amounts $c_{i,0}$ and $c_{i,1}$ are specific to that row. Because σ does not mix any information between different rows, the diffusion it provides is entirely intra-row. The complete linear layer λ is then defined as the composition $\lambda = \rho \circ \sigma$, where ρ is a ShiftRows-like shuffle that moves bits across rows. This design achieves a column branch number of 4 at a cost of 2 XOR operations per bit.

Stoffelen and Daemen [9] formalized the CPM, which generalizes the mixing layer of Keccak-f. A CPM computes the column parity P by XOR-ing all rows, folds it by XOR-ing with a rotated copy of itself, and adds the shifted result to every row. Unlike Ascon's σ , a CPM mixes information from

all rows into every output row, providing inter-row diffusion in a single step. Its cost is 2 XOR/bit and both the differential and linear branch numbers equal 4. Keccak-f's θ step and Xoodoo's θ are well-known instances of the CPM family.

El Hirsch et al. [10] introduced the TCPM at CRYPTO 2023, extending the CPM by maintaining two parity sums P and Q (where Q involves row-dependent cyclic shifts). This raises the bitwise differential branch number to 12 at a cost of $(3 + 1/m)$ XOR/bit. However, the linear branch number remains 4, because the transpose of the TCPM collapses to a CPM-like structure. The permutation Gaston instantiates the TCPM on a 5×64 -bit state; its best-found 3-round differential and linear trail weights exceed those of Ascon. Yet Lei et al. [11] later showed that for four or more rounds, Gaston's linear resistance is weaker than Ascon due to a 2-round iterative linear trail with squared correlation 2^{-48} per period.

Lei et al. [11] identified the root cause of the TCPM's linear weakness and remedied it by introducing row-dependent shifts $b_0, \dots, b_{m-1}$ to the TCPM. The symmetric variant, which sets $b_k = t_k$, has the remarkable property that its transpose is also a symmetric TCPM, so its differential and linear branch histograms are identical. Both branch numbers equal 12, at the cost of 4 XOR/bit. The permutations Gaston-S and SBD are built on the STCPM.

Yu and Liu [12] proposed MRM at IACR ToSC 2025, constructing a linear layer from a sequence of shifted XOR operations (computing $x_i \leftarrow x_j \oplus (x_k \lll c)$) followed by cyclic row shifts. By choosing the structure of the first $r + 1$ operations to produce a dense inverse matrix, MRM ensures that single-S-box output activations require many input S-box activations, preventing efficient 3-round backward trail extensions. With 13 XOR operations for $r = 5$ rows, MRM achieves a differential branch number of 12 at only 2.6 XOR/bit and a linear branch number of 5. The permutation Hsilu uses MRM and outperforms both Ascon and Gaston in hardware gate count and software speed.

C. Our Contributions

Motivated by the gaps identified above, this paper makes the following contributions. We first present a unified treatment of bit-wise and

column-wise differential and linear branch numbers, which serves as the common framework for comparing all constructions (Section II). Building on the analysis of CPM, TCPM, STCPM, and MRM in Section II, we then propose three new lightweight linear layers that occupy previously unoccupied positions in the cost–security space (Section III): Construction 1, a lightweight TCPM variant achieving column branch number 7 at 2.8 XOR/bit; Construction 2, combining Ascon’s σ with 5 MRM-type steps to reach branch number 7 at 3 XOR/bit with an improved linear branch number; and Construction 3, combining σ with 10 MRM-type steps to reach column branch number 17 at 4 XOR/bit, achieving the best known column branch number at this cost among the Ascon- σ +MRM family. Finally, we instantiate these layers in Ascon-like permutations and confirm through MILP-based trail search that the proposed variants achieve at most 35, 41, and 65 active S-boxes over 3 differential rounds, respectively (Section IV).

II. PRELIMINARIES

A. State Model and Notation

We consider an iterated permutation whose round function $R = \gamma \circ \lambda$ operates on a two-dimensional state $A \in \mathbb{F}_2^{m \times n}$ with m rows and n columns. The i -th row is denoted A_i and the bit at position (i, j) is $a_{i,j}$. The non-linear layer γ acts in parallel on each column; the linear layer $\lambda = \rho_{\text{east}} \circ \theta \circ \rho_{\text{west}}$ consists of a mixing layer θ flanked by two ShiftRow-like shuffles. We write $A_i \lll c$ for the left cyclic rotation of A_i by c positions.

B. Differential and Linear Cryptanalysis

Differential cryptanalysis [13] tracks the propagation of input differences. For $f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$, the differential probability is

$$\text{DP}(a, b) = \frac{\#\{x \mid f(x) \oplus f(x \oplus a) = b\}}{2^n}.$$

The differential weight satisfies $\text{DP} = 2^{-w_r}$. An r -round trail T has weight $w_r(T) = \sum_i w_r(t^{i-1}, t^i)$.

Linear cryptanalysis [14] tracks the propagation of linear masks. The correlation of an approximation (α, β) is:

$$C(\alpha, \beta) = \frac{\#\{x \mid \alpha^\top x = \beta^\top f(x)\}}{2^{n-1}} - 1,$$

with correlation weight satisfying $C^2 = 2^{-w_{\text{corr}}}$.

C. Branch Numbers

Let L be a linear mapping on $\mathbb{F}_2^{mn}$. The bit weight $w_b(D)$ counts non-zero bits; the column weight $w_c(D)$ counts columns with at least one non-zero bit.

Definition 1 ([9]). *The differential bit (resp. column) branch number of L is:*

$$B_b^d(L) = \min_{D \neq 0} (w_{\text{bit}}(D) + w_{\text{bit}}(L(D))),$$

$$B_c^d(L) = \min_{D \neq 0} (w_{\text{col}}(D) + w_{\text{col}}(L(D))).$$

The linear counterparts are defined via the transpose L^T :

$$B_b^l(L) = \min_{D \neq 0} (w_{\text{bit}}(D) + w_{\text{bit}}(L^T(D))),$$

$$B_c^l(L) = \min_{D \neq 0} (w_{\text{col}}(D) + w_{\text{col}}(L^T(D))).$$

A larger branch number forces every non-zero difference or mask to activate more bits or S-boxes across one application of L , directly raising the minimum weight of multi-round trails. For 3-round trail analysis, the activity pattern of the inverse is equally important: $N_{\min}^{-1}[d]$ denotes the minimum number of active output S-boxes of L when its inverse receives d active S-boxes as input. A large $N_{\min}^{-1}[1]$ prevents efficient backward extensions of trails through a sparse middle round.

In practice, when linear layers are combined with S-boxes in SPN-based designs, the column-wise diffusion is the most relevant metric, since each column typically corresponds to one S-box. Therefore, branch numbers are usually evaluated at the column level rather than the bit level. In the remainder of this paper, unless stated otherwise, the term branch number refers to the column branch number.

D. Existing Lightweight Linear Layers

We briefly recall the main XOR-and-rotation linear layers introduced in Sections I and II as

baselines for comparison. These layers can be classified into two broad types: intra-row transformations, which mix bits only within each row and require a subsequent row shuffle to diffuse across rows, and inter-row transformations, which directly mix information from multiple rows in a single step. Table IV summarizes their column branch numbers (B_c^d , B_c^l) and costs.

- **Ascon σ (intra-row):** For $0 \leq i < m$, we have

$$\sigma(A)_i = A_i \oplus (A_i \lll c_{i,0}) \oplus (A_i \lll c_{i,1}).$$

The full linear layer $\lambda = \rho \circ \sigma$ adds a ShiftRows-like shuffle ρ . This construction costs 2 XOR operations per bit and achieves $B_c^d = B_c^l = 4$.

- **Column Parity Mixer (CPM) (inter-row):** For $0 \leq i < m$, we have

$$\theta(A)_i = A_i \oplus ((P \oplus (P \lll r)) \lll u),$$

where $P = \bigoplus_{k=0}^{m-1} A_k$. The CPM also costs 2 XOR per bit and offers the same branch numbers $B_c^d = B_c^l = 4$.

- **Twin Column Parity Mixer (TCPM) (inter-row):** For $0 \leq i < m$, we have:

$$\theta(A)_i = A_i \oplus (E \lll u),$$

where

$$E = (P \oplus (P \lll r)) \oplus (Q \oplus (Q \lll s)),$$

$$P = \bigoplus_k A_k, \quad Q = \bigoplus_k (A_k \lll t_k).$$

For $m = 5$ rows, the TCPM costs approximately 3.2 XOR per bit. Its differential column branch number is 12, but its linear column branch number remains 4.

- **Symmetric TCPM (STCPM) (inter-row):**

For $0 \leq i < m$, we have:

$$\theta(A)_i = A_i \oplus ((E \oplus (F \lll b_i)) \lll u),$$

where $F = Q \oplus (Q \lll s)$, with P, Q, E as in TCPM. This symmetry ensures that the transpose L^T is also an STCPM. The STCPM costs 4 XOR per bit and achieves $B_c^d = B_c^l = 12$.

- **Multiple Rows Mixer (MRM) (hybrid):** For $r = 5$ rows, MRM applies t shifted XOR operations followed by row shifts:

$$\theta(x)_i = x_{n_1[i]} \oplus (x_{n_2[i]} \lll m_i), r \leq i < r+t,$$

$$\rho(x)_i = x_i \lll s_i, t \leq i < r+t.$$

The MRM construction costs 2.6 XOR per bit, yields differential branch numbers of 10 or 12, and achieves a linear branch number of 5.

These existing designs define the cost–security landscape that our proposals (Section III) aim to extend.

III. PROPOSED LINEAR LAYERS

A. Design Rationale

The existing linear layers reviewed in Section II reveal several gaps worth addressing. At the low-cost end, both Ascon σ and CPM offer only column branch number 4 despite their efficiency. The TCPM achieves branch number 12 differentially but leaves a linear weakness ($B_c^l = 4$) that translates directly into weak iterative linear trails in Gaston [11]. The STCPM closes this gap but doubles the XOR cost relative to CPM. MRM offers a good trade-off yet does not exceed branch number 12. Notably, no existing design achieves a column branch number above 12 within a budget of 4 XOR/bit.

We therefore propose three constructions that occupy previously unoccupied positions in the cost–security space. Construction 1 targets the gap between CPM/Ascon and MRM, seeking a moderate branch number at near-CPM cost (Section II). Construction 2 refines this idea by incorporating MRM-type inter-row steps after Ascon’s σ to also improve the linear branch number (Section III). Construction 3 pushes further, using more MRM steps at the same cost as STCPM to achieve a branch number substantially higher than any existing design (Section IV). Table IV summarises and compares all constructions.

B. Construction 1: Lightweight TCPM Variant (LTV)

The TCPM’s high differential branch number comes at the price of a four-term combination for E . We ask whether a simpler two-term combination can yield a useful intermediate branch number at lower cost.

The LTV linear layer can be expressed as the composition:

$$\text{LTV} = \rho_{\text{west}} \circ \theta' \circ \rho_{\text{east}}, \quad (1)$$

where ρ_{east} applies row-wise left rotations with offsets $t_{e_i} = [0, 60, 22, 27, 4]$ to the input state Y (the output of the χ layer); ρ_{west} applies row-wise left rotations with offsets $t_{w_i} = [0, 56, 31, 46, 43]$ to produce the next round state X' ; the core transformation θ' acts on a 5×64 state A (the result of ρ_{east}) as follows:

$$\begin{aligned} P &= \bigoplus_{i=0}^4 A_i, Q = \bigoplus_{i=0}^4 (A_i \lll t_i), \\ t_i &= [25, 32, 52, 60, 63], \\ E &= (P \lll 1) \oplus (Q \lll 18), \\ \theta'(A)_i &= A_i \oplus (E \lll 23), 0 \leq i < 5. \end{aligned} \quad (2)$$

In words, after the initial row rotations, two column-wise parities are computed: P is the XOR of the five rows of A , while Q is the XOR of the same rows after each is rotated left by its twist constant t_i . These parities are rotated left by 1 and 18 bits respectively, then XORed together to form E . The resulting E is rotated left by 23 bits and XORed back into every row of A . Finally, ρ_{west} applies the row rotations to obtain the next state X' . Compared to the original TCPM construction, the LTV replaces two folding steps by a single direct XOR of the rotated parities, saving two XOR operations per bit.

Computing P costs $m - 1$ XORs per bit position; computing Q requires a further $m - 1$ XORs; the combination $(P \lll r) \oplus (Q \lll s)$ costs 1 XOR; broadcasting to all m rows costs m XORs per position. The total is $(3m - 1)/m = 14/5 = 2.8$ XOR/bit for $m = 5$. The kernel of LTV consists of all states satisfying $(P \lll r) \oplus (Q \lll s) = \mathbf{0}$. A full algebraic characterisation of this kernel and a proof that $B_c^d(\text{LTV}) = 7$ is tight by experiments are given in Proposition 1. The linear branch number $B_c^l(\text{LTV}) = 4$ is a consequence of the CPM-like structure of the transpose LTV^T , established in Proposition 2: $B_c^d(\text{LTV}) = 7, B_c^l(\text{LTV}) = 4$.

C. Construction 2: Ascon- σ with 5 MRM Steps (Ascon-MRM5)

Applying a small number of MRM-type shifted-XOR operations after Ascon's σ introduces inter-row diffusion, raising the branch number without the full overhead of TCPM or STCPM. With five steps, the linear branch number also improves relative to Construction 1.

TABLE I. ASCON σ ROTATION CONSTANTS $c_{i,0}, c_{i,1}$ AND FINAL ROW-SHIFT OFFSETS s_i (FOR ρ), SHARED BY CONSTRUCTIONS 2 AND 3

i	$c_{i,0}$	$c_{i,1}$	s_i
0	19	28	0
1	61	39	1
2	1	6	3
3	10	17	5
4	7	41	13

Let the initial $5 \times n$ state be $A = (A_0, \dots, A_4)$.

Step 1 (σ). Apply Ascon's row-wise transformation to obtain intermediate rows:

$$x_i = A_i \oplus (A_i \lll c_{i,0}) \oplus (A_i \lll c_{i,1}), 0 \leq i \leq 4.$$

Rotation constants $c_{i,0}, c_{i,1}$ are given in Table I.

Step 2 (MRM phase). Perform $t = 5$ shifted-XOR operations. For $j = 5, \dots, 9$:

$$x_j = x_{p_j} \oplus (x_{q_j} \lll m_j),$$

where $p_j, q_j < j$. Following the MRM design principle of [12], the five operations are chosen so that the inverse submatrix is dense, ensuring a large $N_{\min}^{-1}[1]$. The concrete parameter set is given in Table II.

TABLE II. MRM PARAMETER SET FOR CONSTRUCTION 2 (ASCON-MRM5). EACH STEP COMPUTES $x_{5+j} = x_{p_j} \oplus (x_{q_j} \lll m_j)$ WITH $p_j, q_j < 5 + j$

j	p_j	q_j	m_j
0	1	4	15
1	2	3	34
2	2	1	9
3	0	3	16
4	5	0	27

Step 3 (row shift ρ). Apply a row-specific rotation only to the five newly created rows:

$$y_i = x_{i+5} \lll s_i, \quad 0 \leq i \leq 4,$$

with offsets s_i from Table I. The output state is $(y_0, \dots, y_4)$. The σ step costs 2 XOR/bit. The five shifted-XOR operations produce five output rows, contributing $5/5 = 1$ XOR/bit. Total is 3 XOR/bit. We have $B_c^d(\text{AM5}) = 7, B_c^l(\text{AM5}) = 5$.

The MILP search finds an optimal 2-round differential trail with exactly 7 active S-boxes, matching the lower bound proven in Proposition

1. The activity pattern for the non-linear layer (p_S) and linear layer (p_L) is shown below; each dot is inactive, each '1' is active.

```
Optimal solution found (tolerance 1.00e-04)
Best objective 7.000000000000e+00,
best bound 7.000000000000e+00, gap 0.0000%

.....1.....
.....1.....
.....1.....
.....1.....
.....1.....
-----p_S
.....1.....1.....
.....1.....
.....1.....
.....1.....
-----p_L
.....1.....1.....
.....1.....
.....1.....
.....1.....
-----
Configuration: 3 4
Sum: 7
```

The MILP search yields an optimal 2-round linear trail with exactly 5 active S-boxes.

```
Optimal solution found (tolerance 1.00e-04)
Best objective 5.000000000000e+00,
best bound 5.000000000000e+00, gap 0.0000%

.....1.....1.....
.....1.....1.....
.....1.....1.....
.....1.....1.....
.....1.....1.....
-----p_S
.....1.....1.....1.....
.....1.....
.....1.....
.....1.....
-----p_L
.....1.....1.....1.....
.....1.....
.....1.....
.....1.....
-----
Configuration: 3 2
Total correlation weight: 5
```

D. Construction 3: Ascon- σ with 10 MRM Steps (Ascon-MRM10)

Doubling the MRM phase to $t = 10$ steps achieves a substantially higher branch number at the same cost as STCPM (4 XOR/bit), targeting applications that require a larger security margin. Apply Ascon's σ to obtain $x_0, \dots, x_4$ as before. Then execute $t = 10$ shifted-XOR operations: for $j = 5, \dots, 14$,

$$x_j = x_{p_j} \oplus (x_{q_j} \lll m_j), \quad p_j, q_j < j.$$

Following [10], the first six operations ($j = 5, \dots, 10$, phase p_1) are designed so that the inverse is dense, giving a large $N_{\min}^{-1}[1]$; the remaining four ($j = 11, \dots, 14$, phase p_2) further raise the column branch number. The full parameter set is given in Table III. After the 10 operations we have $x_0, \dots, x_{14}$.

Apply a final row shift ρ to the last five rows:

$$y_i = x_{10+i} \lll s_i, \quad 0 \leq i \leq 4,$$

with offsets s_i from Table I.

The σ step costs 2 XOR/bit; the 10 shifted-XOR operations over 5 output rows cost $10/5 = 2$ XOR/bit. Total is 4 XOR/bit. We have $B_c^d(\text{AM10}) = 17, B_c^l(\text{AM10}) = 8$, surpassing STCPM ($B_c^d = B_c^l = 12$) and MRM ($B_c^d \leq 12, B_c^l = 5$).

The MILP search yields an optimal 2-round differential trail with exactly 17 active S-boxes.

```
Optimal solution found (tolerance 1.00e-04)
Best objective 1.700000000000e+01,
best bound 1.700000000000e+01, gap 0.0000%

1.....
1.....
1.....
.....
-----p_S
.....
1.....
.....
.....
.....
-----p_L
1.....1.....1.....
.....1.....1.....
.....1.....1.....
.....1.....1.....
1.....11.....1.....1.....
-----
Configuration: 1 16
Sum: 17
```

The MILP search yields an optimal 2-round linear trail with exactly 8 active S-boxes.

```
Optimal solution found (tolerance 1.00e-04)
Best objective 8.000000000000e+00,
best bound 8.000000000000e+00, gap 0.0000%

.....1.....
.....1.....
1..1.....1.....
.....1.....
.....11.....
.....11.....
-----p_S
.....
1.....1.....
.....11.....
.....
-----p_L
.....1.....
.....1.....
.....
-----
Configuration: 6 2
Total correlation weight: 8
```

We fix $m = 5$ rows and $n = 64$ columns. For $A \in \mathbb{F}_2^{m \times n}$ we write $P = \bigoplus_{k=0}^{m-1} A_k$ (column parity) and $Q = \bigoplus_{k=0}^{m-1} (A_k \lll t_k)$ (twisted parity), where $t_0, \dots, t_{m-1} \in \{1, \dots, n-1\}$ are distinct. The LTV operator (1) is $\text{LTV}(A)_i = A_i \oplus ((P \oplus Q) \lll r)$. The Ascon-MRM t layer is $\mathcal{L}_t = \rho \circ \text{MRM}_t \circ \sigma$, where MRM_t applies t operations $x_j = x_{p_j} \oplus (x_{q_j} \lll m_j)$.

Proposition 1. [Kernel structure of θ' and branch number of LTV] Let θ' be defined by (2) with twist constants $t_0, \dots, t_{m-1} \in \{1, \dots, n-1\}$ satisfying

$$t_i \neq t_j \text{ for all } i \neq j, t_i \neq 0 \text{ for all } i. \quad (3)$$

Let $s_k = (t_k + 17) \bmod n$ for $k = 0, \dots, m-1$. For the concrete parameter set of Construction 1, $s = [42, 49, 5, 13, 16]$, which is also pairwise distinct and nonzero. Then:

i) The kernel of θ' is

$$\left\{ A \in \mathbb{F}_2^{m \times n} \mid \bigoplus_{k=0}^{m-1} (A_k \oplus (A_k \lll s_k)) = \mathbf{0} \right\}.$$

ii) The minimum column weight of a non-zero element of $\ker(\theta')$ satisfies $w_{\text{col}}(A) \geq 3$ for all $A \in \ker(\theta') \setminus \{\mathbf{0}\}$. Furthermore, exhaustive enumeration over all $s \leq 2$ active column patterns confirms $\ker(\theta') \cap \mathcal{W}_{\leq 2} = \{\mathbf{0}\}$, where $\mathcal{W}_s$ denotes the set of states with at most s active columns. The column differential branch number of the full LTV layer satisfies $B_c^d(\text{LTV}) = 7$.

Proof. (i) Kernel characterisation. $A \in \ker(\theta')$ iff $(P \lll 1) \oplus (Q \lll 18) = \mathbf{0}$, i.e. $P = Q \lll 17$. Writing this out column-wise:

$$\begin{aligned} \bigoplus_{k=0}^{m-1} A_k &= \bigoplus_{k=0}^{m-1} (A_k \lll t_k) \lll 17 \\ &= \bigoplus_{k=0}^{m-1} (A_k \lll t_k + 17), \end{aligned}$$

which rearranges to:

$$\left\{ A \in \mathbb{F}_2^{m \times n} \mid \bigoplus_{k=0}^{m-1} (A_k \oplus (A_k \lll s_k)) = \mathbf{0} \right\}.$$

with $s_k = (t_k + 17) \bmod n$.

(ii) *Minimum column weight in $\ker(\theta')$.* The exhaustive search over all $C(n, 1) \times (2^m - 1) = 1,984$ states with $s = 1$ active column and all $C(n, 2) \times (2^m - 1)^2 = 1,937,376$ states with $s = 2$ active columns confirms that none belongs to $\ker(\theta')$ for the parameter set of Construction 1. The search space for $s = 1$ corresponds to all $(m \times 1)$ -column states with a non-zero m -bit column vector; for $s = 2$ it

TABLE III. MRM PARAMETER SET FOR CONSTRUCTION 3 (ASCON-MRM10). PHASE p_1 : STEPS $j = 5, \dots, 10$ (DENSE INVERSE). PHASE p_2 : STEPS $j = 11, \dots, 14$ (BRANCH-NUMBER RAISING)

Phase p_1				Phase p_2			
j	p_j	q_j	m_j	j	p_j	q_j	m_j
5	1	4	15	11	6	9	61
6	2	3	34	12	11	7	13
7	2	1	9	13	12	8	42
8	0	3	16	14	13	10	43
9	5	0	27				
10	5	4	14				

covers all pairs of column positions with independently chosen non-zero column vectors.

By analyzing the diffusion properties of the round function, the minimum column weight of D and $\text{LTV}(D)$ can be bounded using combinatorial arguments. However, due to the large state space (5 rows $\times$ 64 columns), an exact determination requires computational optimization. We model the two-round propagation as an MILP problem and solve it to optimality using Gurobi (version 13.0). The search certifies that for all non-zero D , $w_{\text{col}}(D) + w_{\text{col}}(\text{LTV}(D)) \geq 7$, and a trail achieving exactly 7 active S-boxes exists. Hence $B_c^d(\text{LTV}) = 7$. The complete MILP source code is provided in the Appendix A.

Proposition 2 (Transpose structure and linear branch number of LTV). For the LTV transformation defined in Construction 1, the linear branch number is $B_c^l(\text{LTV}) = 4$. This result follows from the duality $B_c^l(\text{LTV}) = B_c^d(\text{LTV}^T)$ together with an MILP computation that certifies $B_c^d(\text{LTV}^T) = 4$. The explicit form of $(\theta')^T$ as follows:

$$(\theta')^T(U)_i = U_i \oplus [(\bar{U} \oplus \bar{U} \lll -s_i) \lll -(r+\alpha)].$$

Proof. We first compute the adjoint of θ' with respect to the standard inner product $\langle U, A \rangle = \bigoplus_{i,j} u_{i,j} a_{i,j}$. Recall that $\theta'(A)_i = A_i \oplus (E \lll r)$ with $E = (P \lll \alpha) \oplus (Q \lll \beta)$, where $r = 23$, $\alpha = 1$, $\beta = 18$ for Construction 1, and $P = \bigoplus_k A_k$, $Q = \bigoplus_k (A_k \lll t_k)$. Using $\langle a, b \lll c \rangle = \langle a \lll -c, b \rangle$ and expanding, we obtain

$$\begin{aligned}
\langle U, \theta'(A) \rangle &= \sum \langle U_i, A_i \rangle \oplus \langle \bar{U} \lll -(r + \alpha), P \rangle \\
&\oplus \langle \bar{U} \lll -(r + \beta), Q \rangle \\
&= \sum_i \langle U_i, A_i \rangle \oplus \sum_k \langle \bar{U} \lll -(r + \alpha), A_k \rangle \\
&\oplus \sum_k \langle \bar{U} \lll -(r + \beta + t_k), A_k \rangle.
\end{aligned}$$

Collecting the coefficient of $\langle \cdot, A_i \rangle$ gives

$$(\theta')^T(U)_i = U_i \oplus \bar{U} \lll -(r + \alpha) \oplus \bar{U} \lll -(r + \beta + t_i).$$

Writing $s_i = t_i + (\beta - \alpha) \pmod{n}$, this becomes

$$(\theta')^T(U)_i = U_i \oplus [(\bar{U} \oplus \bar{U} \lll -s_i) \lll -(r + \alpha)].$$

For the concrete parameters ($n = 64$, $r + \alpha = 24$, s_i as given by the t_i and $\Delta = 17$), the map $(\theta')^T$ has the same structure as θ' and satisfies the same non-degeneracy conditions.

Since $\text{LTV} = \rho_{\text{west}} \circ \theta' \circ \rho_{\text{east}}$ and the row-rotation layers are orthogonal ($\rho^T = \rho^{-1}$), we have:

$$\text{LTV}^T = \rho_{\text{east}}^{-1} \circ (\theta')^T \circ \rho_{\text{west}}^{-1}.$$

Thus, the linear branch number of LTV^T is evaluated by the same MILP model (Section IV) adapted to $(\theta')^T$. Running the model to optimality with Gurobi yields $B_c^d(\text{LTV}^T) = 4$ directly. By the standard duality $B_c^l(L) = B_c^d(L^T)$ [11], we conclude $B_c^l(\text{LTV}) = 4$.

Corollary 1 (Iterative linear trail in Ascon-C1). Any Ascon-like permutation instantiated with LTV as its linear layer admits a 2-round iterative linear trail with squared correlation 2^{-12} per period.

IV. APPLICATION TO ASCON-LIKE PERMUTATIONS

A. Overview of Existing Permutations

We briefly recall the trail bounds of the Ascon-like permutations that serve as reference points. Ascon [8] has $B_c^d = B_c^l = 4$ with minimum 3-round differential and linear trail weights of 40 and 28 [10]. Gaston [10] achieves $B_c^d = 12$ but $B_c^l = 4$; its best-found 3-round weights are ≤ 106 (differential) and 34 (linear),

TABLE IV. EXISTING AND PROPOSED LINEAR LAYERS (5×64 -BIT STATE). B_c^d : DIFFERENTIAL COLUMN BRANCH NUMBER; B_c^l : LINEAR COLUMN BRANCH NUMBER. PROPOSED CONSTRUCTIONS ARE MARKED WITH $\dagger$

Linear Layer	XOR/bit	B_c^d	B_c^l
Ascon σ	2.0	4	4
CPM (Keccak / Xoodoo)	2.0	4	4
TCPM (Gaston)	3.2	12	4
STCPM (Gaston-S)	4.0	12	12
MRM (Hsilu)	2.6	10	5
MRM variant	3.0	14	5
C1: LTV †	2.8	7	4
C2: Ascon-MRM5 †	3.0	7	5
C3: Ascon-MRM10 †	4.0	17	8

yet for four or more rounds its linear resistance falls below Ascon's due to a 2-round iterative trail with squared correlation 2^{-48} [11]. Gaston-S [9] corrects this by raising both branch numbers to 12, giving 3-round weights ≤ 148 and ≤ 141 . Hsilu [12] uses MRM with $B_c^d = 10$, achieving 3-round weights in $[48, 84]$ and $[38, 52]$ despite its lower cost of 2.6 XOR/bit. These bounds are collected in Table V.

TABLE V. TRAIL WEIGHT BOUNDS FOR EXISTING ASCON-LIKE PERMUTATIONS. INTERVAL $[a, b]$: PROVEN LOWER BOUND a , BEST-FOUND UPPER BOUND b

Permutation	Differential		Linear	
	B_c^d	3-rd wt.	B_c^l	3-rd wt.
Ascon	4	40	4	28
Gaston	12	≤ 106	4	34
Gaston-S	12	≤ 148	12	≤ 141
Hsilu	10	$[48, 84]$	5	$[38, 52]$

B. Ascon-Like Permutations with Proposed Linear Layers

We instantiate three Ascon-like permutations: Ascon-C1, Ascon-C2, Ascon-C3 by replacing Ascon's σ with Constructions 1, 2, and 3 respectively, while retaining the χ non-linear layer and the ι round constant addition.

The 2-round lower bounds follow directly from the column branch numbers: a trail with a and b

active S-boxes in rounds 1 and 2 satisfies $a + b \geq B_c^d$, giving minima of 7, 7, and 17 for Ascon-C1, Ascon-C2, and Ascon-C3 respectively, compared to 4 for Ascon and 12 for Gaston.

For 3-round bounds, we search for the best differential trail using MILP, fixing the number of active S-boxes in the middle round and solving the resulting model. The 2-round bounds are exact because the branch number directly gives the minimum: the MILP model can prove infeasibility for objective values below B_c^d , and the MILP-found trail certifies the exact optimum. For 3-round trails, the MILP objective encodes an upper bound on the minimum trail weight: the solver finds the best trail it can within the given time and search space, but cannot in general certify that no lighter trail exists, because the number of candidate patterns grows exponentially with rounds and the MILP formulation encodes only a relaxation of the actual count. Proven 3-round lower bounds would require either a complete enumeration (computationally infeasible at this state size) or additional algebraic structure arguments that we leave to future work. The 3-round figures reported below are therefore best-found upper bounds, consistent with the notation $\leq$ used throughout. The results are:

- Ascon-C1 (LTV, 2.8 XOR/bit): at most 35 active S-boxes over 3 rounds.
- Ascon-C2 (Ascon-MRM5, 3 XOR/bit): at most 41 active S-boxes over 3 rounds.
- Ascon-C3 (Ascon-MRM10, 4 XOR/bit): at most 65 active S-boxes over 3 rounds.

Table VI summarizes these results alongside the existing permutations. Ascon-C1 and Ascon-C2 improve on Ascon (15 active S-boxes in 2 rounds vs. 4, and 35 and 41 in 3 rounds vs. 15) at costs of only 2.8 and 3 XOR/bit. Ascon-C2 additionally avoids the linear weakness of Ascon-C1 ($B_c^l = 5$ vs. $B_c^l = 4$). Ascon-C3 reaches 65 active S-boxes over 3 rounds at the same cost as Gaston-S, substantially exceeding Gaston-S's bound of ≤ 58 , making it attractive for high-security applications such as sponge constructions with a large capacity.

V. DISCUSSION

The three proposed constructions each address a specific gap in the existing design space. Construction 1 (LTV) requires only 0.8 XOR/bit more than Ascon or CPM, yet raises

TABLE VI. MINIMUM ACTIVE S-BOXES OVER 2 AND 3 ROUNDS. †: PROPOSED VARIANTS. 3-ROUND COLUMN: UPPER BOUND (BEST-FOUND DIFFERENTIAL TRAIL)

Permutation	XOR/bit	2-round	3-round
Ascon	2.0	4	15
Gaston	3.2	12	≤ 50
Gaston-S	4.0	12	≤ 58
Hsilu	2.6	10	≤ 41
Ascon-C1†	2.8	7	≤ 35
Ascon-C2†	3.0	7	≤ 41
Ascon-C3†	4.0	17	≤ 65

the column branch number from 4 to 7 and brings the 3-round S-box count from 15 to at most 35, a meaningful improvement for a negligible cost increase. Its main limitation is the unchanged linear branch number of 4, inherited from the CPM-like structure of $P \oplus Q$, which may lead to iterative linear trails analogous to those in Gaston. Construction 2 (Ascon-MRM5) resolves this by adding inter-row MRM steps that raise B_c^l to 5 at the cost of one additional XOR per bit; it also achieves 41 active S-boxes in 3 rounds. Construction 3 (Ascon-MRM10) is the strongest result: at 4 XOR/bit it reaches $B_c^d = 17$ and 65 active S-boxes, outcomes that no existing 4 XOR/bit design approaches, making it the best known construction at this cost level within the Ascon- σ +MRM family. All other cryptographic properties (algebraic degree growth, division property, linear hull correlations) depend on the Ascon S-box and remain unchanged; therefore their analysis is outside the scope of this paper, which focuses solely on linear diffusion.

VI. CONCLUSION

In this paper, we analyzed the branch number properties of existing lightweight linear layers such as CPM, TCPM, STCPM, and MRM. Moreover, we proposed three new constructions targeting different cost-security trade-offs. Construction 1 (LTV) achieves column branch number 7 at 2.8 XOR/bit. Construction 2 (Ascon-MRM5) matches this branch number at 3 XOR/bit while also raising the linear branch number to 5. Construction 3 (Ascon-MRM10) achieves column branch number 17 at 4 XOR/bit, achieving the best known column branch number

at this cost level among all designs in the Ascon- σ +MRM family evaluated. MILP-based trail search on the corresponding Ascon-like permutations confirms these improvements, with Ascon-C1, Ascon-C2, and Ascon-C3 achieving at most 35, 41, and 65 active S-boxes over 3 differential rounds respectively, compared to 15 for standard Ascon and ≤ 58 for Gaston-S at the same budget.

ACKNOWLEDGMENT

This work was funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 02/2025/TN. This work was also partially supported by the Academy of Cryptography Techniques.

APPENDIX

This appendix provides the complete C++ source code that implements the MILP models for Constructions 1, 2, and 3 using the Gurobi optimizer. The code computes the minimal number of active S-boxes over a given number of rounds, thereby determining the column branch numbers and optimal differential trails reported in Sections III and IV.

```
//
// =====
// Gurobi models for the proposed linear layers
// Constructions 1 (LTV), 2 (Ascon-MRM5), 3 (Ascon-MRM10)
#include <gurobi_c++.h>
#include <sstream>
string itos(int i) {stringstream s; s << i; return s.str();}
}
#define N 320
GRBEnv env = GRBEnv();
// Inequalities for Ascon Sbox
const int INEQ[50][11] = {{4, 9, 3, 9, 5, -2, 2, -1, 0, -6,
0},
...
{2, -1, -1, -1, 1, 0, 0, -2, -2, 2, 5}};
//
// =====
// Construction 1-LTV
//
// =====
int ltv_model(int threadNumber, int rounds);
int ltv_model(int threadNumber, int rounds) {
    int i, j, r;
    try {
        env.set(GRB_IntParam_LogToConsole, 1);
        env.set(GRB_IntParam_Threads, threadNumber);
        GRBModel model = GRBModel(env);
        // ---- Main variables ----
        vector<vector<GRBVar>> X(rounds, vector<
GRBVar>(320));
        vector<vector<GRBVar>> Y(rounds-1, vector<
GRBVar>(320));
        vector<vector<GRBVar>> D(rounds, vector<
GRBVar>(64));
        // ---- Auxiliary variables for the LTV
        linear layer ----
        vector<vector<GRBVar>> T1(rounds-1, vector<
GRBVar>(320));
        vector<vector<GRBVar>> T2(rounds-1, vector<
GRBVar>(320));
        vector<vector<GRBVar>> P(rounds-1, vector<
GRBVar>(64));
        vector<vector<GRBVar>> Q(rounds-1, vector<
GRBVar>(64));
        vector<vector<GRBVar>> E(rounds-1, vector<
GRBVar>(64));
        vector<vector<GRBVar>> F(rounds-1, vector<
GRBVar>(320));
```

```
// Integer auxiliary variables for XOR
constraints
    vector<vector<GRBVar>> U(rounds-1, vector<
GRBVar>(64)); // used for P
    vector<vector<GRBVar>> V(rounds-1, vector<
GRBVar>(64)); // used for Q
    vector<vector<GRBVar>> B(rounds-1, vector<
GRBVar>(64)); // used for E
    // Local XOR2add function (creates an
    auxiliary variable automatically)
    auto XOR2add = [&](GRBVar a, GRBVar b,
GRBVar c) {
        GRBVar aux = model.addVar(0, 2, 0,
GRB_INTEGER);
        model.addConstr(a + b + c == 2 * aux
);
    };
    // Constants for LTV (as defined)
    const int EE[5] = {0, 60, 22, 27, 4}; //
rho_east
    const int TT[5] = {25, 32, 52, 60, 63};
    const int RR = 1;
    const int SS = 18;
    const int UU = 23;
    const int WW[5] = {0, 56, 31, 46, 43}; //
rho_west
    // Initialize variables
    for (r = 0; r < rounds; r++) {
        for (i = 0; i < 320; i++)
            X[r][i] = model.addVar(0, 1, 0,
GRB_BINARY, "X_"+itos(r)+"_"+itos(i));
        for (i = 0; i < 64; i++)
            D[r][i] = model.addVar(0, 1, 0,
GRB_BINARY, "D_"+itos(r)+"_"+itos(i));
        for (r = 0; r < rounds-1; r++) {
            for (i = 0; i < 320; i++) {
                Y[r][i] = model.addVar(0, 1,
0, GRB_BINARY, "Y_"+itos(r)+"_"+itos(i));
                T1[r][i] = model.addVar(0,
1, 0, GRB_BINARY, "T1_"+itos(r)+"_"+itos(i));
                T2[r][i] = model.addVar(0,
1, 0, GRB_BINARY, "T2_"+itos(r)+"_"+itos(i));
                F[r][i] = model.addVar(0, 1,
0, GRB_BINARY, "F_"+itos(r)+"_"+itos(i));
                for (i = 0; i < 64; i++) {
                    P[r][i] = model.addVar(0, 1,
0, GRB_BINARY, "P_"+itos(r)+"_"+itos(i));
                    Q[r][i] = model.addVar(0, 1,
0, GRB_BINARY, "Q_"+itos(r)+"_"+itos(i));
                    E[r][i] = model.addVar(0, 1,
0, GRB_BINARY, "E_"+itos(r)+"_"+itos(i));
                    U[r][i] = model.addVar(0, 3,
0, GRB_INTEGER, "U_"+itos(r)+"_"+itos(i));
                    V[r][i] = model.addVar(0, 3,
0, GRB_INTEGER, "V_"+itos(r)+"_"+itos(i));
                    B[r][i] = model.addVar(0, 2,
0, GRB_INTEGER, "B_"+itos(r)+"_"+itos(i));
                }
            }
        }
    }
    // ===== Constraints over rounds
    for (r = 0; r < rounds-1; r++) {
        // --- S-box (50 inequalities) ---
        for (i = 0; i < 64; i++) {
            model.addConstr(X[r][i] + X[
r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] >= D[r][i]);
            model.addConstr(X[r][i] + X[
r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] <= 5*D[r][i]);
            model.addConstr(Y[r][i] + Y[
r][64+i] + Y[r][128+i] + Y[r][192+i] + Y[r][256+i] >= D[r][i]);
            model.addConstr(Y[r][i] + Y[
r][64+i] + Y[r][128+i] + Y[r][192+i] + Y[r][256+i] <= 5*D[r][i]);
            for (j = 0; j < 50; j++) {
                GRBLinearExpr tt = 0;
                for (int k = 0; k <
5; k++) tt += INEQ[j][k] * X[r][64*k + i];
                for (int k = 0; k <
5; k++) tt += INEQ[j][5+k] * Y[r][64*k + i];
                tt += INEQ[j][10];
                model.addConstr(tt
>= 0);
            }
        }
        // --- Step 1: rho_east (Y -> T1)
        for (i = 0; i < 64; i++) T1[r][i]
= Y[r][((i + EE[0]) % 64) + 0];
        for (i = 0; i < 64; i++) T1[r][64+i]
= Y[r][((i + EE[1]) % 64) + 64];
        for (i = 0; i < 64; i++) T1[r][128+i]
= Y[r][((i + EE[2]) % 64) + 128];
        for (i = 0; i < 64; i++) T1[r][192+i]
= Y[r][((i + EE[3]) % 64) + 192];
        for (i = 0; i < 64; i++) T1[r][256+i]
= Y[r][((i + EE[4]) % 64) + 256];
        // --- Step 2: P = XOR of the 5 rows
```

```

of T1 (same column) ---
    for (i = 0; i < 64; i++) {
        model.addConstr(T1[r][i] +
T1[r][64+i] + T1[r][128+i] + T1[r][192+i] + T1[r][256+i] + P[r
][i] == 2 * U[r][i]);
    }
// --- Step 3: Apply twist rotations
and compute Q ---
    for (i = 0; i < 64; i++) T2[r][i]
= T1[r][((i + TT[0]) % 64) + 0];
    for (i = 0; i < 64; i++) T2[r][64+i]
= T1[r][((i + TT[1]) % 64) + 64];
    for (i = 0; i < 64; i++) T2[r][128+i]
= T1[r][((i + TT[2]) % 64) + 128];
    for (i = 0; i < 64; i++) T2[r][192+i]
= T1[r][((i + TT[3]) % 64) + 192];
    for (i = 0; i < 64; i++) T2[r][256+i]
= T1[r][((i + TT[4]) % 64) + 256];

    for (i = 0; i < 64; i++) {
        model.addConstr(T2[r][i] +
T2[r][64+i] + T2[r][128+i] + T2[r][192+i] + T2[r][256+i] + Q[r
][i] == 2 * V[r][i]);
    }
// --- Step 4: E = (P <<< RR) XOR (Q
<<< SS) ---
    for (i = 0; i < 64; i++) {
        model.addConstr(P[r][i + RR
) % 64] + Q[r][i + SS] % 64] + E[r][i] == 2 * B[r][i]);
    }
// --- Step 5: F = T1 XOR (E <<< UU)
(broadcast) ---
    for (i = 0; i < 64; i++) {
        for (j = 0; j < 5; j++) {
            XOR2add(T1[r][64*j +
i], E[r][i + UU] % 64, F[r][64*j +
i]);
        }
// --- Step 6: rho_west (F -> X[r
+1]) ---
        for (i = 0; i < 64; i++) model.
== F[r][((i + WW[0]) % 64) + 0];
        for (i = 0; i < 64; i++) model.
== F[r][((i + WW[1]) % 64) + 64];
        for (i = 0; i < 64; i++) model.
== F[r][((i + WW[2]) % 64) + 128];
        for (i = 0; i < 64; i++) model.
== F[r][((i + WW[3]) % 64) + 192];
        for (i = 0; i < 64; i++) model.
== F[r][((i + WW[4]) % 64) + 256];
    }
// ===== Last round (only chi)
=====
    r = rounds-1;
    for (i = 0; i < 64; i++) {
        model.addConstr(X[r][i] + X[r][64+i]
+ X[r][128+i] + X[r][192+i] + X[r][256+i] >= D[r][i]);
        model.addConstr(X[r][i] + X[r][64+i]
+ X[r][128+i] + X[r][192+i] + X[r][256+i] <= 5*D[r][i]);
    }
// ===== Objective function =====
vector<GRBLinExpr> KS(rounds);
GRBLinExpr KSS;
    for (j = 0; j < rounds; j++) {
        for (i = 0; i < 64; i++) KS[j] += D[
j][i];
        KSS += KS[j];
    }
    model.addConstr(KS[0] >= 1);
    model.setObjective(KSS, GRB_MINIMIZE);
    model.optimize();
// ===== Output results =====
    for (r = 0; r < rounds-1; r++) {
        // print X[r]
        for (int row = 0; row < 5; row++) {
            for (int col = 0; col < 64;
col++)
                cout << (round(X[r][row*64+
col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
            cout << endl;
        }
        cout << "
-----
p_S" << endl;

        // print Y[r]
        for (int row = 0; row < 5; row++) {
            for (int col = 0; col < 64;
col++)
                cout << (round(Y[r][row*64+
col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
            cout << endl;
        }
        cout << "
-----
p_L" << endl;

        // print X[rounds-1]
        for (int row = 0; row < 5; row++) {
            for (int col = 0; col < 64; col++)
                cout << (round(X[rounds-1][row*64+
col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
            cout << endl;
        }
    }
}

```

```

cout << "
-----
" << endl;

    int AS[rounds];
    for (i = 0; i < rounds; i++) AS[i] = round(
KS[i].getValue());
    cout << "Configuration:";
    for (i = 0; i < rounds; i++) cout << AS[i]
<< " ";
    cout << endl;
    cout << "Sum:" << round(model.getObjective()
.getValue()) << endl;
    if (model.get(GRB_IntAttr_Status) ==
GRB_INFEASIBLE) return -1;
    else if (model.get(GRB_IntAttr_Status) ==
GRB_OPTIMAL) return 1;
    else return -2;
}
catch (GRBException e) {
    cerr << "Errorcode=" << e.getErrorCode() <<
endl;
    cerr << e.getMessage() << endl;
}
catch (...) {
    cerr << "Exceptionduringoptimization" <<
endl;
}
return -1;
}
//
// Construction 2-Ascon-MRM5
//
=====
int mrm5_model(int threadNumber, int rounds);
int mrm5_model(int threadNumber, int rounds) {
    int i, j, r;
    try {
        env.set(GRB_IntParam_LogToConsole, 1);
        env.set(GRB_IntParam_Threads, threadNumber);
        GRBModel model = GRBModel(env);
        // ---- Main variables ----
        vector<vector<GRBVar>> X(rounds, vector<
GRBVar>(320));
        vector<vector<GRBVar>> Y(rounds-1, vector<
GRBVar>(320));
        vector<vector<GRBVar>> D(rounds, vector<
GRBVar>(64));
        // ---- Auxiliary variables for the MRMS
linear layer ----
        const int M = 5;
        const int NOPS = 5;
        const int NTOT = M + NOPS; // 10
        // xv[r][k][i] : k = 0..9, i = 0..63
        vector<vector<vector<GRBVar>>> xv(rounds-1,
vector<vector<GRBVar>>(NTOT, vector<GRBVar
>(64)));
        // Integer auxiliary variables for XOR (
using XOR2add lambda)
        // Local XOR2add function
        auto XOR2add = [&](GRBVar a, GRBVar b,
GRBVar c) {
            GRBVar aux = model.addVar(0, 2, 0,
GRB_INTEGER);
            model.addConstr(a + b + c == 2 * aux
);
        };
        // Initialize variables
        for (r = 0; r < rounds; r++) {
            for (i = 0; i < 320; i++)
                X[r][i] = model.addVar(0, 1, 0,
GRB_BINARY, "X_"+itos(r)+"_"+itos(i));
            for (i = 0; i < 64; i++)
                D[r][i] = model.addVar(0, 1, 0,
GRB_BINARY, "D_"+itos(r)+"_"+itos(i));
        }
        for (r = 0; r < rounds-1; r++) {
            for (i = 0; i < 320; i++)
                Y[r][i] = model.addVar(0, 1, 0,
GRB_BINARY, "Y_"+itos(r)+"_"+itos(i));
            for (int k = 0; k < NTOT; k++)
                xv[r][k][i] = model.addVar(0, 1, 0,
GRB_BINARY,
                    "xv_"+itos(r)+"_"+itos(k)+"_"+itos(i)
));
        }
        // Constants for Ascon-MRM5
        const int CC0[5] = {19, 61, 1, 10, 7};
        const int CC1[5] = {28, 39, 6, 17, 41};
        const int MRMS_P[5] = {1, 2, 2, 0, 5};
        const int MRMS_Q[5] = {4, 3, 1, 3, 0};
        const int MRMS_M[5] = {15, 34, 9, 16, 27};
        const int WW[5] = {0, 1, 3, 5, 13}; //
Ascon's rho_west
// ===== Constraints over rounds
=====
        for (r = 0; r < rounds-1; r++) {
            // --- S-box (68 inequalities) ---
            for (i = 0; i < 64; i++) {
                model.addConstr(X[r][i] + X[
r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] >= D[r][i])
;
            }
        }
    }
}

```

```

        model.addConstr(X[r][i] + X[r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] <= 5*D[r][i]);
        model.addConstr(Y[r][i] + Y[r][64+i] + Y[r][128+i] + Y[r][192+i] + Y[r][256+i] >= D[r][i]);
        model.addConstr(Y[r][i] + Y[r][64+i] + Y[r][128+i] + Y[r][192+i] + Y[r][256+i] <= 5*D[r][i]);

        for (j = 0; j < 68; j++) {
            GRBLinExpr tt = 0;
            for (int k = 0; k < 5; k++) tt += INEQ[j][k] * X[r][64*k + i];
            for (int k = 0; k < 5; k++) tt += INEQ[j][5+k] * Y[r][64*k + i];
            tt += INEQ[j][10];
            model.addConstr(tt >= 0);
        }
    }
    // --- Step 1: sigma (XOR of 3 bits)
    -> xv[r][0..4] ---
    column i:
        // tmp = Y[row][i] XOR Y[row][i+CC0]
        // xv[row][i] = tmp XOR Y[row][i+CC1]
        for (int row = 0; row < 5; row++) {
            for (i = 0; i < 64; i++) {
                GRBVar tmp = model.addVar(0, 1, 0, GRB_BINARY, "tmp_"+itos(row)+"_"+itos(i));
                // tmp = Y[row][i]
                XOR2add(Y[r][row*64 + i],
                    Y[r][row*64 + ((i + CC0[row]) % 64)],
                    tmp);
                // xv = tmp XOR Y[row][i+CC1]
                XOR2add(tmp,
                    Y[r][row*64 + ((i + CC1[row]) % 64)],
                    xv[r][row][i]);
            }
        }
    // --- Step 2: 5 MRM operations ---
    // xv[r][5+k][i] = xv[r][p][i] XOR
    for (int k = 0; k < NOPS; k++) {
        int p = MRM5_P[k];
        int q = MRM5_Q[k];
        int m = MRM5_M[k];
        for (i = 0; i < 64; i++) {
            XOR2add(xv[r][p][i],
                xv[r][q][(i+m) % 64],
                xv[r][5+k][i]);
        }
    }
    // --- Step 3: rho_west (rotate the
    last rows) ---
    // X[r+1][row][i] = xv[r][5+row][(i + WW[row]) % 64]
    for (int row = 0; row < 5; row++) {
        for (i = 0; i < 64; i++) {
            model.addConstr(X[r+1][row*64 + i] == xv[r][5 + row][(i + WW[row]) % 64]);
        }
    }
    // ===== Last round (only chi)
    =====
    r = rounds-1;
    for (i = 0; i < 64; i++) {
        model.addConstr(X[r][i] + X[r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] >= D[r][i]);
        model.addConstr(X[r][i] + X[r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] <= 5*D[r][i]);
    }
    // ===== Objective function =====
    vector<GRBLinExpr> KS(rounds);
    GRBLinExpr KSS;
    for (j = 0; j < rounds; j++) {
        for (i = 0; i < 64; i++) KS[j] += D[j][i];
        KSS += KS[j];
    }
    model.addConstr(KS[0] >= 1);
    model.setObjective(KSS, GRB_MINIMIZE);
    model.optimize();
    // ===== Output results =====
    for (r = 0; r < rounds-1; r++) {
        for (int row = 0; row < 5; row++) {
            for (int col = 0; col < 64; col++)
                cout << (round(X[r][row*64+col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
            cout << endl;
        }
    }
}

```

```

        cout << "
-----
p_S" << endl;
        for (int row = 0; row < 5; row++) {
            for (int col = 0; col < 64; col++)
                cout << (round(Y[r][row*64+col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
            cout << endl;
        }
        cout << "
-----
p_L" << endl;
        for (int row = 0; row < 5; row++) {
            for (int col = 0; col < 64; col++)
                cout << (round(X[rounds-1][row*64+col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
            cout << endl;
        }
        cout << "
-----
" << endl;
        int AS[rounds];
        for (i = 0; i < rounds; i++) AS[i] = round(KS[i].getValue());
        cout << "Configuration:";
        for (i = 0; i < rounds; i++) cout << AS[i] << " ";
        cout << endl;
        cout << "Sum:" << round(model.getObjective().getValue()) << endl;
        if (model.get(GRB_IntAttr_Status) == GRB_INFEASIBLE) return -1;
        else if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL) return 1;
        else return -2;
    }
    catch (GRBException e) {
        cerr << "Errorcode=" << e.getErrorCode() << endl;
        cerr << e.getMessage() << endl;
    }
    catch (...) {
        cerr << "Exceptionduringoptimization" << endl;
    }
    return -1;
}
//
=====
// Construction 3-Ascon-MRM10
//
int mrm10_model(int threadNumber, int rounds);
int mrm10_model(int threadNumber, int rounds) {
    int i, j, r;
    try {
        env.set(GRB_IntParam_LogToConsole, 1);
        env.set(GRB_IntParam_Threads, threadNumber);
        GRBModel model = GRBModel(env);
        // ---- Main variables ----
        vector<vector<GRBVar>> X(rounds, vector<GRBVar>(320));
        vector<vector<GRBVar>> Y(rounds-1, vector<GRBVar>(320));
        vector<vector<GRBVar>> D(rounds, vector<GRBVar>(64));
        // ---- Auxiliary variables for the MRM10 linear layer ----
        const int M = 5;
        const int NOPS = 10;
        const int NTOT = M + NOPS; // 15
        vector<vector<vector<GRBVar>>> xv(rounds-1, vector<vector<GRBVar>>(NTOT, vector<GRBVar>>(64)));
        auto XOR2add = [&](GRBVar a, GRBVar b, GRBVar c) {
            GRBVar aux = model.addVar(0, 2, 0, GRB_INTEGER);
            model.addConstr(a + b + c == 2 * aux);
        };
        // Initialize
        for (r = 0; r < rounds; r++) {
            for (i = 0; i < 320; i++)
                X[r][i] = model.addVar(0, 1, 0, GRB_BINARY, "X_"+itos(r)+"_"+itos(i));
            for (i = 0; i < 64; i++)
                D[r][i] = model.addVar(0, 1, 0, GRB_BINARY, "D_"+itos(r)+"_"+itos(i));
        }
        for (r = 0; r < rounds-1; r++) {
            for (i = 0; i < 320; i++)
                Y[r][i] = model.addVar(0, 1, 0, GRB_BINARY, "Y_"+itos(r)+"_"+itos(i));
            for (int k = 0; k < NTOT; k++)
                for (i = 0; i < 64; i++)
                    xv[r][k][i] = model.addVar(0, 1, 0, GRB_BINARY, "xv_"+itos(r)+"_"+itos(k)+"_"+itos(i));
        }
    }
}

```

```

// Constants for Ascon-MRM10
const int CC0[5] = {19, 61, 1, 10, 7};
const int CC1[5] = {28, 39, 6, 17, 41};
const int MRM10_P[10] =
{1,2,2,0,5,5,6,11,12,13};
const int MRM10_Q[10] =
{4,3,1,3,0,4,9,7,8,10};
const int MRM10_M[10] =
{15,34,9,16,27,14,61,13,42,43};
const int WW[5] = {0, 1, 3, 5, 13};
for (r = 0; r < rounds-1; r++) {
// --- S-box (68 inequalities) ---
for (i = 0; i < 64; i++) {
model.addConstr(X[r][i] + X[
r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] >= D[r][i])
;
model.addConstr(X[r][i] + X[
r][64+i] + X[r][128+i] + X[r][192+i] + X[r][256+i] <= 5*D[r][i
]);
model.addConstr(Y[r][i] + Y[
r][64+i] + Y[r][128+i] + Y[r][192+i] + Y[r][256+i] >= D[r][i])
;
model.addConstr(Y[r][i] + Y[
r][64+i] + Y[r][128+i] + Y[r][192+i] + Y[r][256+i] <= 5*D[r][i
]);

for (j = 0; j < 68; j++) {
GRBLinExpr tt = 0;
for (int k = 0; k <
5; k++) tt += INEQ[j][k] * X[r][64*k + i];
for (int k = 0; k <
5; k++) tt += INEQ[j][5+k] * Y[r][64*k + i];
tt += INEQ[j][10];
model.addConstr(tt
>= 0);
}
}
// --- Step 1: sigma (XOR of 3 bits)
-> xv[r][0..4] ---
for (int row = 0; row < 5; row++) {
for (i = 0; i < 64; i++) {
GRBVar tmp = model.
XOR2add(Y[r][row*64
+ i],
Y[r][row*64 + ((i +
tmp);
XOR2add(tmp,
Y[r][row*64 + ((i +
xv[r][row][i]);
}
}
// --- Step 2: 10 MRM operations ---
for (int k = 0; k < NOPS; k++) {
int p = MRM10_P[k];
int q = MRM10_Q[k];
int m = MRM10_M[k];
for (i = 0; i < 64; i++) {
XOR2add(xv[r][p][i],
xv[r][q][i + m] %
64],
xv[r][5 + k][i]);
}
}
// --- Step 3: rho_west (take rows
10..14) ---
// X[r+1][row][i] = xv[r][10+row][(i
+ WW[row]) % 64]
for (int row = 0; row < 5; row++) {
for (i = 0; i < 64; i++) {
model.addConstr(X[r
+1][row*64 + i] == xv[r][10 + row][(i + WW[row]) % 64]);
}
}
// Last round
r = rounds-1;
for (i = 0; i < 64; i++) {
model.addConstr(X[r][i] + X[r][64+i]
+ X[r][128+i] + X[r][192+i] + X[r][256+i] >= D[r][i]);
model.addConstr(X[r][i] + X[r][64+i]
+ X[r][128+i] + X[r][192+i] + X[r][256+i] <= 5*D[r][i]);
}
// Objective
vector<GRBLinExpr> KS(rounds);
GRBLinExpr KSS;
for (j = 0; j < rounds; j++) {
for (i = 0; i < 64; i++) KS[j] += D[
j][i];
KSS += KS[j];
}
model.addConstr(KS[0] >= 1);
model.setObjective(KSS, GRB_MINIMIZE);
model.optimize();
// Output
for (r = 0; r < rounds-1; r++) {
for (int row = 0; row < 5; row++) {
for (int col = 0; col < 64;
col++)
cout << (round(X[r][row*64+

```

```

col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".";
cout << endl;
}
cout << "
-----
p_S" << endl;
for (int row = 0; row < 5; row++) {
for (int col = 0; col < 64;
col++)
cout << (round(Y[r][row*64+
col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
cout << endl;
}
cout << "
-----
p_L" << endl;
for (int row = 0; row < 5; row++) {
for (int col = 0; col < 64; col++)
cout << (round(X[rounds-1][row*64+
col].get(GRB_DoubleAttr_Xn)) == 1 ? "1" : ".");
cout << endl;
}
cout << "
-----
" << endl;
int AS[rounds];
for (i = 0; i < rounds; i++) AS[i] = round(
KS[i].getValue());
cout << "Configuration:";
for (i = 0; i < rounds; i++) cout << AS[i]
<< " ";
cout << endl;
cout << "Sum:" << round(model.getObjective()
.getValue()) << endl;
if (model.get(GRB_IntAttr_Status) ==
GRB_INFEASIBLE) return -1;
else if (model.get(GRB_IntAttr_Status) ==
GRB_OPTIMAL) return 1;
else return -2;
}
catch (GRBException e) {
cerr << "Errorcode=" << e.getErrorCode() <<
endl;
cerr << e.getMessage() << endl;
}
catch (...) {
cerr << "Exceptionduringoptimization" <<
endl;
}
return -1;
}
}

```

Listing 1: Gurobi models for Constructions 1–3

REFERENCES

- [1] National Institute of Standards and Technology, “Lightweight Cryptography Standardization” (2023), Access time: 12/06/2026, *Computer Security Resource Center (CSRC)*, <https://csrc.nist.gov/projects/lightweight-cryptography>.
- [2] N. V. Long, H. D. Linh and L. Q. Dat, “The quantum circuit construction for S-boxes without ancilla qubits: A more detail analysis,” *Journal of Science and Technology on Information Security*, vol. 24, no. 1, pp. 44–55, 2024. DOI: 10.54654/isj.v1i24.1031.
- [3] T. T. Luong, T. M. Phuong, N. V. Long and N. N. Khanh, “Enhancing SPN ciphers: Dynamic substitution-key addition layers via binary block circulant matrices,” *Journal of Science and Technology on Information Security*, vol. 24, no. 1, pp. 12-29, 2025. DOI: 10.54654/isj.v1i24.1100.
- [4] S. Duval and G. Leurent, “MDS matrices with lightweight circuits,” *IACR Transactions on Symmetric Cryptology*, vol. 2018, no. 2, pp. 48–78, 2018. DOI: 10.13154/tosc.v2018.i2.48-78.
- [5] T. Kranz, G. Leander, K. Stoffelen and F. Wiemer, “Shorter linear straight-line programs for MDS matrices,” *IACR Transactions on Symmetric*

- Cryptology*, vol. 2017, no. 4, pp. 188–211, 2017. DOI: 10.13154/tosc.v2017.i4.188-211.
- [6] G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, “Keccak specifications,” NIST SHA-3 submission document, 2009.
- [7] J. Daemen, S. Hoffert, G. Van Assche and R. Van Keer, “The design of Xoodoo and Xooff,” *IACR Transactions on Symmetric Cryptology*, vol. 2018, no. 4, pp. 1–38, 2018. DOI:10.13154/tosc.v2018.i4.
- [8] C. Dobraunig, M. Eichlseder, F. Mendel and M. Schl  ffer, “Ascon v1.2: Lightweight authenticated encryption and hashing,” *Journal of Cryptology*, vol. 34, no. 3, 2021. DOI:10.1007/s00145-021-09398-9.
- [9] K. Stoffelen and J. Daemen, “Column parity mixers,” *IACR Transactions on Symmetric Cryptology*, vol. 2018, no. 1, pp. 126–159, 2018. DOI: 10.13154/tosc.v2018.i1.126-159.
- [10] S. El Hirsch, J. Daemen, R. Rohit and R. H. Makarim, “Twin column parity mixers and Gaston,” in *Advances in Cryptology – CRYPTO 2023*, Lecture Notes in Computer Science, vol. 14083, Cham, Switzerland: Springer, pp. 475–506, 2023. DOI: 10.1007/978-3-031-38548-3_16.
- [11] H. Lei, R. Rohit, G. Liu, J. He, M. Rachidi, K. Jia, K. Hu and M. Wang, “Symmetric twin column parity mixers and their applications,” *IACR Transactions on Symmetric Cryptology*, vol. 2024, no. 4, pp. 1–37, 2024. DOI: 10.46586/tosc.v2024.i4.1-37.
- [12] X. Yu and M. Liu, “Multiple rows mixers and Hsilu,” *IACR Transactions on Symmetric Cryptology*, vol. 2025, no. 2, pp. 166–191, 2025. DOI: 10.46586/tosc.v2025.i2.166-191.
- [13] E. Biham and A. Shamir, “Differential cryptanalysis of DES-like cryptosystems,” *Journal of Cryptology*, vol. 4, no. 1, pp. 3–72, 1991. DOI: 10.1007/BF00630563.
- [14] M. Matsui, “Linear cryptanalysis method for DES cipher,” in *Advances in Cryptology – EUROCRYPT’93*, Lecture Notes in Computer Science, vol. 765, Berlin, Germany: Springer, pp. 386–397, 1994. DOI: 10.1007/3-540-48285-7_33.

ABOUT THE AUTHORS



Hoàng Đình Linh

Workplace: Institute of Cryptography Science and Technology, Vietnam Government Information Security Commission. He is also a member of the Advanced Cryptography Research Lab (ACR Lab), a joint research initiative between the Vietnam

Institute for Advanced Study in Mathematics (VIASM) and the Institute of Cryptography Science and Technology.

Email: hoangdinhlinh@bcy.gov.vn

Education: In 2014, he graduated from the Advanced Mathematics Program, obtaining a Bachelor’s degree in Mathematics from the VNU University of Science, Vietnam National University, Hanoi. In 2021, he received a Master’s degree in Applied Mathematics from the VNU

University of Science, Vietnam National University, Hanoi. He is currently a Ph.D. candidate in Cryptographic Engineering at the Academy of Cryptography Techniques, where his research focuses on symmetric cryptography, lightweight cryptography, and cryptographic permutations.

Recent research direction: Design and security evaluation of symmetric cryptographic algorithms, lightweight cryptography, cryptographic permutations, statistical testing of randomness, random number generators for cryptographic applications, and post-quantum cryptography.

Tên tác giả: Hoàng Đình Linh

Cơ quan làm việc: Viện Khoa học - Công nghệ mật mã, Ban Cơ yếu Chính phủ. Anh đồng thời là thành viên của Phòng Nghiên cứu Mật mã Tiên tiến (Advanced Cryptography Research Lab - ACR Lab), một mô hình hợp tác nghiên cứu giữa Viện Nghiên cứu cao cấp về Toán (VIASM) và Viện Khoa học - Công nghệ mật mã.

Email: hoangdinhlinh@bcy.gov.vn.

Quá trình đào tạo: Năm 2014 tốt nghiệp Chương trình Cử nhân nhân Tiên tiến Toán học tại Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội, nhận bằng Cử nhân Toán học. Năm 2021 nhận bằng Thạc sĩ Toán ứng dụng tại Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội. Hiện nay anh đang là nghiên cứu sinh chuyên ngành Kỹ thuật mật mã tại Học viện Kỹ thuật mật mã.

Hướng nghiên cứu hiện nay: Thiết kế và đánh giá tính an toàn của các thuật toán mã hóa đối xứng, mật mã hạng nhẹ, hoán vị mật mã, các phương pháp kiểm định, đánh giá tính ngẫu nhiên, bộ sinh số ngẫu nhiên, mật mã hậu lượng tử.



Trần Thị Lương

Workplace: Academy of Cryptography Techniques, Vietnam.

Education: She received her M.S. degree and Ph.D. degree in Cryptographic Engineering. In 2026, she was awarded the title of Associate Professor in Information Security.

Recent research direction: Her research focuses on cryptography, coding theory, information security, lightweight cryptography, and post-quantum cryptography. She has extensive experience in cryptographic research and has actively contributed to the organization of scientific events in the fields of cryptography and cybersecurity. She served as a co-chair of special sessions at KSE 2023, VICRIS 2024, and VICRIS 2025.

Tên tác giả: Trần Thị Lương

Cơ quan làm việc: Học viện Kỹ thuật mật mã, Việt Nam

Quá trình đào tạo: Bà nhận bằng Thạc sĩ và Tiến sĩ chuyên ngành Kỹ thuật mật mã. Năm 2026, bà được phong học hàm Phó Giáo sư ngành An toàn thông tin.

Hướng nghiên cứu hiện nay: Mật mã học, lý thuyết mã và an toàn thông tin. Bà có nhiều kinh nghiệm nghiên cứu trong lĩnh vực mật mã và an toàn thông tin, đồng thời tham gia tổ chức nhiều hội nghị khoa học quốc tế. Bà từng đảm nhiệm vai trò đồng chủ trì các phiên chuyên đề tại KSE 2023, VICRIS 2024 và VICRIS 2025.



Nguyen Van Long

Workplace: Academy of
Cryptography Techniques, Vietnam.

Email: nvlong.bcy@gmail.com

Education: He received his Engineer's
degree in Information Security of
Telecommunication Systems from
the FSO Academy, Russian Federation,

in 2008, and his Ph.D. degree from the FSO Academy,
Russian Federation, in 2015.

Recent research direction: Cryptography, Coding theory
and Information Security.

Tên tác giả: **Nguyễn Văn Long**

Cơ quan làm việc: Học viện Kỹ thuật mật mã, Việt Nam

Email: nvlong.bcy@gmail.com

Quá trình đào tạo: Năm 2008 tốt nghiệp Kỹ sư chuyên ngành
An toàn thông tin các hệ thống viễn thông tại Học viện FSO,
Liên bang Nga. Năm 2015 nhận bằng Tiến sĩ tại Học viện
FSO, Liên bang Nga.

Hướng nghiên cứu hiện nay: Mật mã học, lý thuyết mã và an
toàn thông tin.