# Sycon: A New Milestone in Designing ASCON-like Permutations

Kalikinkar Mandal[1], Dhiman Saha[2], Sumanta Sarkar[3] and Yosuke Todo[4]

[1] University of New Brunswick, Canada, kmandal@unb.ca
[2] IIT Bhilai, India, dhiman@iitbhilai.ac.in
[3] TCS Innovation Labs, India, sumanta.sarkar1@tcs.com
[4] NTT Secure Platform Laboratories, Japan,todo.yosuke@gmail.com

**Abstract.** ASCON is one of the elegant designs of authenticated encryption with associated data (AEAD) that was selected as the first choice for lightweight applications in the CAESAR competition, which also has been submitted to NIST lightweight cryptography standardization. ASCON has been in the literature for a while, however, there has been no successful AEAD which is secure at the same time lighter than ASCON. In this article, we have overcome the challenge of constructing a permutation that is lighter than the ASCON permutation while ensuring a similar performance, and based on which we achieve a more lightweight AEAD which we call Sycon. Extensive security analysis of Sycon confirms that it provides the same level of security as that of ASCON. Our hardware implementation result shows that the Sycon permutation has 5.35% reduced area, compared to the ASCON permutation. This leads to a remarkable area reduction for Sycon AEAD which is about 14.95% as compared to ASCON AEAD. We regard the Sycon as a new milestone as it is the lightest among all the AEADs belonging to the ASCON family.

**Keywords:** ASCON, Lightweight cryptography, AEAD, Hardware implementation, Diffusion, S-box

## 1 Introduction

With the proliferation of the Internet of Things (IoT), a new branch of cryptography called lightweight cryptography has emerged. Lightweight cryptography aims to provide security to resource constrained IoT devices. Over the years, numerous lightweight symmetric-key ciphers have been proposed, and interests have been shown by both industry and academia. Standardization efforts are on to incorporate lightweight cryptography into practice. For example, lightweight block ciphers PRESENT [17] and CLEFIA [36] have been standardized by ISO/IEC 29192. Later, the CAESAR competition was launched to standardize Authenticated Encryption with Associated Data (AEAD) algorithms [1]. At the end of the competition, ASCON was selected as the first choice for lightweight applications. Notably, National Institute of Standards and Technology (NIST) also has taken up an initiative to standardize lightweight cryptographic algorithms (NIST-LWC) [28]. In this regard, they announced a call for submissions of lightweight AEAD and hashing algorithms. Currently, in the second round, there are 32 submissions competing in the NIST-LWC to get promoted to the next round [30].

In recent years, several lightweight block ciphers have been proposed, for example, GIFT [9], SKINNY [11], and MIDORI [6], to name a few. Apart from block ciphers or AEAD constructions, literature has also been enriched with lightweight modes of operation for authenticated encryption such as BEETLE [18] and SUNDAE [7]. In fact, several AEAD schemes in the NIST-LWC competition have combined existing

lightweight block ciphers or permutations with lightweight modes. For instance, SUNDAE-GIFT [8] combines GIFT and the SUNDAE mode, and PHOTON-BEETLE [10] combines the PHOTON permutation [24] and the BEETLE mode. One can refer to [29] for more such examples.
Sponge mode and its variants [14, 13, 15] are popular choices as they provide the flexibility of constructing both AEAD and hash functionalities. ASCON built a lightweight permutation of 320 bits and used it in a MonkeyDuplex type mode. This permutation is based on a substitution-permutation network (SPN). It is nicely designed to provide a great flexibility in both hardware and software. ASCON is also competing in the NIST-LWC competition and currently is in the second round.
Other contemporary designs that were inspired by ASCON are DRYGASCON [32], SHAMASH [31] and Sycon v1.0 [34]. DRYGASCON attempted to generalize ASCON with a permutation-equivalent S-box and a slightly modified diffusion layer. SHAMASH also followed the same design principle by replacing ASCON's S-box and diffusion layer.

**Our contribution.** We revisit our construction Sycon v1.0 which was a round 1 candidate in the NIST-LWC competition. After a careful analysis we observe that it lacks in good diffusion property. We find a 3-round differential trail with 10 active S-boxes which is 1 less than what was claimed in the submission document. Our analysis also finds a 4-round differential with only 21 active S-boxes (see Figure 9), which is much less than the heuristic upper bound of 51 as provided for 4-round Sycon v1.0. In this article, we design a new permutation that provides concrete security guarantees while rendering low hardware cost and name it Sycon that supersedes Sycon v1.0.

- **Design rationale.** To design a permutation that is lighter than ASCON, we need to construct a lighter S-box layer and a diffusion layer. First, we generate many new lightweight S-boxes having the same cryptographic properties as that of ASCON. The lightest one we obtain has a hardware cost of 27.34 GE (UMC 65nm technology), which is even lighter than the ASCON S-box. Next, we investigate the diffusion layer of the form $X \oplus (X \lll u) \oplus (X \lll v)$ as used in ASCON. We observe that restricting our choice of rotation constants to $(u, v) = (u, 2u)$ significantly reduces the hardware cost. However, this choice of rotation constants decreases the number of active S-boxes. To compensate that, we introduce additional word rotations which are chosen in accordance with the S-box. This additional word rotation does not introduce any overhead in hardware. As a result, we obtain a diffusion layer which elevates the number of active S-boxes significantly. We apply the MonkeyDuplex [13] mode to construct AEADs and sponge mode [15] for hashing from the Sycon permutation.
- **Security analysis.** We perform an extensive security analysis of the Sycon permutation against powerful cryptanalytic attacks such as differential, linear, integral and differential-linear attacks. We apply the mixed integer linear programming (MILP) tool [26, 37, 3] to determine the number of active S-boxes. For the 3-round permutation, Sycon has 16 differentially and linearly active S-boxes, whereas ASCON has 15 and 13 active S-boxes, respectively. Our analysis shows that the 12-round Sycon permutation is resistant to the state-of-the-art cryptanalytic attacks, thereby achieving the same level of security as of ASCON.
- **Optimized hardware implementation.** We implement the Sycon permutation, two AEAD modes and one hash mode in both ASIC (UMC 65nm) and FPGA (Kintex-7) to demonstrate their efficiency, and provide an extensive evaluation and comparison results. The area for the Sycon permutation is 5303 GE as compared to 5603 GE for ASCON. The areas for two AEAD algorithms and one hash algorithm are 6436, 6537, and 6038 GE, respectively. Comprehensive performance results including power and energy are furnished in Section 6.
- **Comparative analysis.** We provide a detailed comparison with those similar designs that are not broken yet (see Table 1). To have a fair comparison,

we implement ASCON in the same technology. Our result shows that the Sycon permutation has an area improvement of about 5.35% over the ASCON permutation. When the AEAD and hashing of Sycon and ASCON with a rate of 64 are compared, the area improvements are about 14.95% and 4.45%, respectively. As regards the maximum frequency reported by the ASIC synthesis tool, Sycon achieves 1.826 GHz against 1.242 GHz by ASCON which translates to an improvement of 48.3% and is primarily attributed to the mode of operation. In the process of making the hardware extremely lightweight, we loose the software speed of the Sycon permutation approximately 10% slower than that of ASCON. This results in about 10% reduction in software speed when we compare both ASCON and Sycon modes for the rate of width 64 and large messages. We emphasize that the hardware saving is absolute and universal, so this leads to a significant cumulative savings in the manufacturing cost. Therefore, we believe that gaining such a hardware cost by sacrificing as little as 10% in software speed-up is a reasonable trade-off.

Overall, Sycon is well-placed for lightweight applications. Our construction shows how to push the boundary of hardware cost of a permutation that belongs to the ASCON family.

Table 1: Comparing Sycon with other contemporary ciphers of comparable state-sizes. Comparisons are made with those similar designs that are not broken yet.

| Primitive | State-Size | FPGA | | | | ASIC | | | References |
|---|---|---|---|---|---|---|---|---|---|
| | | Technology | Slice | | $F_{Max}$ | Technology | AREA | $F_{Max}$ | |
| | | | LUT | Registers | (MHz) | | (GE) | (GHz) | |
| Sycon-AEAD-64 | 320 | Kintex-7 | **754** | 328 | 354 | UMC 65nm | **6436** | 1.826 | **This Paper** |
| Sycon-AEAD-96 | 320 | | **734** | 328 | 429 | | **6537** | 1.826 | |
| Sycon-Hash-64 | 320 | | **646** | 326 | 483 | | **6038** | 1.884 | |
| ASCON-128 | 320 | | 1055 | 328 | 303 | | 7567 | 1.242 | |
| ASCON-128a | 320 | | 1107 | 328 | 312 | | 7614 | 1,246 | |
| ASCON-Hash | 320 | | 629 | 326 | 429 | | 6319 | 1.884 | |
| DRYGASCON | 320 | Zynq-7000 | 1647 | 625 | 153.8 | −− | −− | −− | [32] |
| Gimli† | 384 | Spartan-6 | 587 | 394 | 202 | ST 28nm | 8097 | 0.939 | [12] |
| ACE | 320 | Spartan 6 | 1272 | 365 | 123 | ST Micro 65nm | 4250 | 0.720 | [2] |
| ISAP-A-128 | 320 | −− | −− | −− | −− | UMC 130nm | 12780 | 0.169 | [20] |

† Results are for the permutation.

# 2 Preliminaries

The Substitution-Permutation Network (SPN) has been widely followed in constructing block ciphers. For example, AES [19] is built on the same principle. Similar approach can be followed to construct cryptographic permutations. Basically in an SPN, there is one substitution layer which is nonlinear and a diffusion layer which is linear. The substitution layer applies S-boxes in parallel that induce the confusion property, and the linear diffusion layer is largely responsible for diffusing the plaintext into the cipher. Using the *wide-trail* strategy of AES [19], one can measure the strength of SPN ciphers against two basic attacks such as differential cryptanalysis [16] and linear cryptanalysis [27]. This basically counts the number of *active* S-boxes for some rounds.

**Differential and linear cryptanalysis.** Let $\mathbb{F}_2^m$ be the vector space formed by the $2^m$ binary $m$-tuples. An S-box of $m$-bit is a mapping $\mathcal{S} : \mathbb{F}_2^m \to \mathbb{F}_2^m$. Differential and linear cryptanalysis are the basic attacks that the designer needs to take care of. To resist the differential cryptanalysis, the S-box should have low differential uniformity defined as follows. Let $\mathcal{D}_S(\delta, \Delta) = \{\#x : \mathcal{S}(x) \oplus \mathcal{S}(x \oplus \delta) = \Delta\}$. The

*differential uniformity* is defined as $\mathsf{DU}(\mathcal{S}) = \max_{\delta\neq 0,\Delta\neq 0} \mathcal{D}_{\mathcal{S}}(\delta,\Delta)$. An S-box is called $k$-differential uniform if $\mathsf{DU}(\mathcal{S}) = k$. On the other hand, in a linear cryptanalysis, the attacker exploits the probabilistic linear relations, called *linear approximations*. The linear approximation table (LAT) keeps the record of biases of the relations of the form $\oplus_{i=0}^{m-1} a_i x_i = \oplus_{i=0}^{m-1} b_i y_i$, where $(x_0,\ldots,x_{m-1})$ and $(y_0,\ldots,y_{m-1})$ are the input and output of an S-box respectively, and $(a_0,\ldots,a_{m-1}) \in \mathbb{F}_2^m$ and $(b_0,\ldots,b_{m-1}) \in \mathbb{F}_2^m$ are called input and output mask respectively. The maximum probability for all the nonzero input and output mask pairs is called the *linear probability* of $\mathcal{S}$ denoted by $\mathcal{L}_{\mathcal{S}}$.
For any $\alpha,\beta \in \mathbb{F}_2^m$, the correlation coefficient of S-box $\mathcal{S}$ with respect to $(\alpha,\beta)$ is given by $\mathcal{C}_{\mathcal{S}}(\alpha,\beta) = \sum_{x\in\mathbb{F}_2^m} (-1)^{\beta\cdot\mathcal{S}(x)+\alpha\cdot x}$.

**Differential and linear branch numbers.** For an $m$-bit S-box $\mathcal{S}$, its differential branch number is defined as

$$\mathsf{DBN}(\mathcal{S}) = \min_{x,y\in\mathbb{F}_2^m,\, x\neq y} \{wt(x\oplus y) + wt(\mathcal{S}(x)\oplus\mathcal{S}(y))\}$$

where $wt(z)$ is the Hamming weight of $z \in \mathbb{F}_2^m$.
For an $m$-bit S-box $\mathcal{S}$, its linear branch number, denoted as $\mathsf{LBN}(\mathcal{S})$, is defined as

$$\mathsf{LBN}(\mathcal{S}) = \min_{\alpha,\beta\in\mathbb{F}_2^m\setminus\{0\},\, \mathcal{C}_{\mathcal{S}}(\alpha,\beta)\neq 0} \{wt(\alpha) + wt(\beta)\}.$$

# 3 Specification of Sycon

In this section, we provide a complete specification of the Sycon permutation and briefly describe its AEAD and hash modes. We design Sycon to be small in hardware with high throughput and to be fast on software platforms.

## 3.1 The Sycon Permutation

Sycon permutation is an iterative permutation of 320 bits whose round function is based on SPN. To produce the output of a 320-bit input, the input is loaded into the internal state of the permutation and then the round function is applied iteratively.

**The State.** The internal state is viewed as a $5 \times 64$ array of bits (see Fig. 1). We denote the 320-bit state by $\mathbf{s} = \mathbf{s}_4\|\mathbf{s}_3\|\mathbf{s}_2\|\mathbf{s}_1\|\mathbf{s}_0$, which is by concatenating the rows where each $\mathbf{s}_i = (s_{i,63}, s_{i,62}, \cdots, s_{i,0}) \in \{0,1\}^{64}$ is a row. Therefore, a 320-bit state is of the form $\mathbf{s} = s_{4,63}, s_{4,62}, \cdots\cdots, s_{0,1}, s_{0,0}$, where $s_{4,63}$ is the MSB and $s_{0,0}$ is the LSB of the state.

**The Round Function.** The round function ($R$) of the Sycon permutation consists of a sequence of three distinct transformations: SBox (SB), SubBlockDiffusion (SD) and AddRoundConst (RC), i.e., $R = \mathtt{RC} \circ \mathtt{SD} \circ \mathtt{SB}$. The $\rho$-round permutation, denoted by $\Pi^\rho$, is constructed as

$$\Pi^\rho = \underbrace{R \circ \cdots \circ R}_{\rho \text{ times}}.$$

Fig. 10 presents a block diagram of the round function whose components are detailed below.

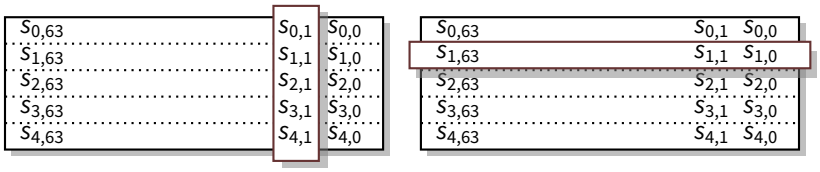
SBox (SB). The substitution box is a nonlinear transformation that provides confusion in the permutation. A 5-bit S-box of Sycon v1.0 is applied to each column of the state. Table 2 provides the decimal representation of the S-box. The SB layer is applied to the state $\mathbf{s}$ by applying 64 S-boxes in parallel on $\mathbf{b}_i = (s_{4,i}, s_{3,i}, s_{2,i}, s_{1,i}, s_{0,i})$, $0 \le i \le 63$. Symbolically, $\mathbf{s}$ is updated as

$$\mathbf{s} \leftarrow \mathbf{SB}(\mathbf{s}) = S[\mathbf{b}_{63}]S[\mathbf{b}_{62}]\cdots S[\mathbf{b}_0],$$

where $\mathbf{b}_i \leftarrow S(\mathbf{b}_i), 0 \le i \le 63$.

Table 2: The 5-bit S-box [34]

| $x$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $S[x]$ | 8 | 19 | 30 | 7 | 6 | 25 | 16 | 13 | 22 | 15 | 3 | 24 | 17 | 12 | 4 | 27 |
| $x$ | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 19 | 30 | 31 |
| $S[x]$ | 11 | 0 | 29 | 20 | 1 | 14 | 23 | 26 | 28 | 21 | 9 | 2 | 31 | 18 | 10 | 5 |




(a) SB layer operation (b) SD layer operation

Figure 1: A $5 \times 64$ array view of the state and the SB and SD layers

SubBlockDiffusion (SD). The diffusion layer is a linear transformation that is applied independently on five 64-bit subblocks $\mathbf{s}_i, 0 \leq i \leq 4$. The linear transformation has the form $L(x) = (x \oplus (x \lll u) \oplus (x \lll 2u)) \lll t$ where $0 \leq u, t \leq 63$ are positive integers, and $\lll$ is the left rotation operation. Note that our linear transformations are different from ASCON's ones.

$$\begin{aligned}
\mathbf{s}_0 &\leftarrow (\mathbf{s}_0 \oplus (\mathbf{s}_0 \lll 59) \oplus (\mathbf{s}_0 \lll 54)) \lll 40 \\
\mathbf{s}_1 &\leftarrow (\mathbf{s}_1 \oplus (\mathbf{s}_1 \lll 55) \oplus (\mathbf{s}_1 \lll 46)) \lll 32 \\
\mathbf{s}_2 &\leftarrow (\mathbf{s}_2 \oplus (\mathbf{s}_2 \lll 33) \oplus (\mathbf{s}_2 \lll 2)) \lll 16 \\
\mathbf{s}_3 &\leftarrow (\mathbf{s}_3 \oplus (\mathbf{s}_3 \lll 21) \oplus (\mathbf{s}_3 \lll 42)) \lll 56 \\
\mathbf{s}_4 &\leftarrow (\mathbf{s}_4 \oplus (\mathbf{s}_4 \lll 13) \oplus (\mathbf{s}_4 \lll 26)).
\end{aligned}$$

AddRoundConst (RC). We generate the round constants using a 4-bit LFSR defined by the primitive polynomial $x^4 + x + 1$ over $\mathbb{F}_2$. The LFSR state is denoted as $\mathbf{rc} = (rc_{i+3}, rc_{i+2}, rc_{i+1}, rc_i)$ where $rc_{i+4} = rc_i \oplus rc_{i+1}$. We start with the initial state $\mathbf{rc} = (0, 1, 0, 1)$ to generate $\rho = 12$ round constants where each state of the LFSR is served as distinct constants. A 4-bit LFSR with state $(rc_3, rc_2, rc_1, rc_0)$ is converted to a byte as $(0, 0, 0, 0, rc_{i+3}, rc_{i+2}, rc_{i+1}, rc_i)$. For example, the 4-bit LFSR state $(0, 1, 0, 1)$ is converted to a byte as $(0, 0, 0, 0, 0, 1, 0, 1) =$ `0x05`, and then a 64-bit round constant is constructed by concatenating the resultant `0x05` with a 56-bit constant value as `0xaaaaaaaaaaaaaaa`$\|$`0x05` = `0xaaaaaaaaaaaaaa05`. The round constants are given in Table 3.

Table 3: The round constants for the Sycon permutation

| Rnd | Constants | Rnd | Constants |
|---|---|---|---|
| 0 | `0xaaaaaaaaaaaaaa05` | 6 | `0xaaaaaaaaaaaaaa03` |
| 1 | `0xaaaaaaaaaaaaaa0a` | 7 | `0xaaaaaaaaaaaaaa01` |
| 2 | `0xaaaaaaaaaaaaaa0d` | 8 | `0xaaaaaaaaaaaaaa08` |
| 3 | `0xaaaaaaaaaaaaaa0e` | 9 | `0xaaaaaaaaaaaaaa04` |
| 4 | `0xaaaaaaaaaaaaaa0f` | 10 | `0xaaaaaaaaaaaaaa02` |
| 5 | `0xaaaaaaaaaaaaaa07` | 11 | `0xaaaaaaaaaaaaaa09` |

**Number of Rounds.** The number of rounds $\rho$ for the Sycon full permutation is set to 12. The full round as well as a reduced-round Sycon permutation is used to construct AEAD instances.

## 3.2 The Sycon Modes

We propose two instances of AEAD based on the Sycon permutation in the MonkeyDuplex mode [13] for resource-constrained applications. More specifically, we use a mode, as shown in Figure 2, which is an amalgamation of NORX [5] and

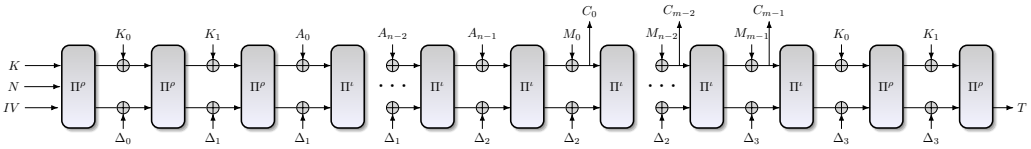



Figure 2: The authentication operation of Sycon for $\iota = 6$ and 7 for Sycon-AEAD-64 and Sycon-AEAD-96, respectively. The domain separators are: $\Delta_0 = 0^c$ for initialization, $\Delta_1 = 100\|0^{c-3}$ for AD processing, $\Delta_2 = 010\|0^{c-3}$ for message, $\Delta_3 = 001\|0^{c-3}$ for tag generations. If the AD is empty, $\Delta_1$ is replaced by $\Delta_2$.

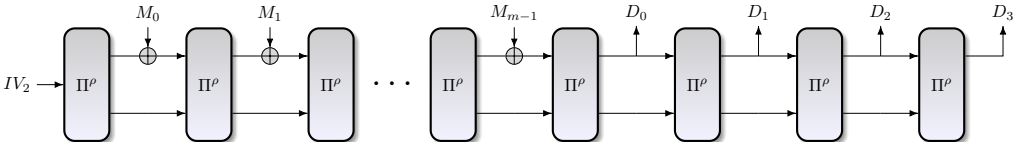



Figure 3: The hash mode for the Sycon permutation.

sLiSCP [4]. Like the sLiSCP mode, the key is absorbed through the rate, instead of the capacity as done in the ASCON mode, which saves the hardware cost of 128 XORs. We use the identical domain separation constants as used in the NORX mode. Table 4 presents the parameters and the corresponding security level for both AEAD instances. Each instance is determined by the key length, the nonce length, the tag length, the rate length, and an initial vector (IV) (see Table 12). For both instances, we limit processing the number of plaintexts and associated data blocks to $2^{64}$ per key. We use the Sycon permutation in the unified sponge mode [15] to construct a hash function (see Figure 3). The parameters for the hash function and the security claims are provided in Table 5. We present a detailed description of the AEAD and hash modes in Appendix A.

Table 4: Recommended parameters and security claims for Sycon AEAD instances

| **Instances** | **Key** | **Nonce** | **Tag** | **Capacity** | **Rate** | **Rounds** | | **Confiden** | **Integ** | **Authen** |
|---|---|---|---|---|---|---|---|---|---|---|
| | $(\kappa)$ | $(n)$ | $(\tau)$ | $(c)$ | $(r)$ | $\rho$ | $\iota$ | **-tiality** | **-rity** | **-ticity** |
| Sycon-AEAD-64 | 128 | 128 | 128 | 256 | 64 | 12 | 6 | 128 | 128 | 128 |
| Sycon-AEAD-96 | 128 | 128 | 128 | 224 | 96 | 12 | 7 | 128 | 128 | 128 |

# 4 Design Rationale for Sycon

In this section, we discuss the rationale behind choosing each component of the Sycon permutation, AEAD and hash modes. We focus on choosing the components so that they lead to a low hardware implementation cost while maintaining a high-speed in software.

## 4.1 Choice of the S-boxes

In our design, we decide to use a $5 \times 5$ S-box with good cryptographic properties along with differential and linear branch number 3 to ensure faster diffusion. ASCON used such an S-box that is affine equivalent to Keccak S-box [15]. One possible attempt to find a hardware efficient S-box is to exhaustively search the affine equivalent class of ASCON S-box. However, this exhaustive search is difficult to complete due to a high computational complexity as the number of binary nonsingular $5 \times 5$ matrices is $\approx 2^{24}$. On the other hand, a more systematic method was presented in [33] for finding $5 \times 5$ and $6 \times 6$ S-boxes with differential and linear branch number 3 using

Table 5: Recommended parameters for the Sycon hash

| Algorithm | Digest ($\ell_h$) | Rate ($r$) | Capacity ($c$) | Rounds ($\rho$) | Preimage resistance | 2nd preimage resistance | Collision resistance |
|---|---|---|---|---|---|---|---|
| Sycon-Hash-64 | 256 | 64 | 256 | 12 | 192 | 128 | 128 |

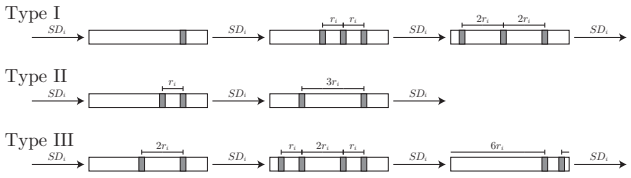



Figure 4: Three diffusion properties whose number of active bits is small.

the relationship between 1-resilient Boolean functions and S-boxes with linear branch number 3. We revisit the latter approach and generate quadratic $5 \times 5$ S-boxes with differential and linear branch number 3 along with nonlinearity 8 and differential uniformity 8. In this process, we obtain many such S-boxes and their hardware cost range from 27.34 GE to 48.44 GE. Note that the cost of ASCON S-box is 28.12 GE. Thus, we choose the S-box with the cost 27.34 GE which gives the first upper hand over ASCON. This S-box is also efficient in software which is evident from its bit-slice form as given in Appendix B.
The cryptographic properties of the chosen S-box are as follows: differential uniformity 8, nonlinearity 8, algebraic degree 2, differential branch number 3, and linear branch number 3.

## 4.2 Choice of the SubBlockDiffusion Layer

Sycon inherits some design philosophies from ASCON, but there are important differences in the choice of the linear diffusion layer SD. The linear diffusion layer of ASCON consists of five different linear diffusion layers, and the $i$th layer denoted by $SD_i$ is represented as

$$SD_i \ : \ Y_i = X_i \oplus (X_i \lll r_i) \oplus (X_i \lll r_i').$$

ASCON chose the parameters $(r_i, r_i')$ such that it achieves high security.
In the SD layer of Sycon, we restrict ourselves to choose $r_i' = 2r_i$. Then, since we can have many shareable XORs, this choice enables us to reduce the area size for hardware implementation. On the other hand, this restriction is not welcome when we focus on the diffusion performance. For example, when the diffusion of differences is taken into account, it allows efficient differential transitions such as the Type II diffusion in Fig. 4. To compensate for this security degradation, additional rotations are applied and it increases the number of active S-boxes in both differential and linear characteristics. Moreover, to minimize the cost on specific software platforms, byte-wise rotations are used. In summary, the $SD_i$ of Sycon is represented as

$$SD_i \ : \ Y_i = (X_i \oplus (X_i \lll r_i) \oplus (X_i \lll 2r_i)) \lll v_i,$$

where $v_i$ is a multiple of eight. We first introduce five criteria to pick the rotation numbers $r_4, \ldots, r_0$. Then, in each candidate of the rotation numbers $r_4, \ldots, r_0$ satisfying five criteria, we pick the rotation numbers $v_4, \ldots, v_0$ such that the numbers of active S-boxes are 16 in both 3-round differential and linear trails.

**Choice of $r_i$.** Clearly, using the same rotation numbers is not preferable. Therefore, the number of choices for five rotation numbers, $r_4, \ldots, r_0$, is $\binom{64}{5}$.
First of all, we choose $r_i$ from odd numbers because $SD_i$ cannot diffuse the whole of 64 bits when $r_i$ is even number.

**Criterion 1.** *For any* $i \in \{4,3,2,1,0\}$, $r_i$ *is odd number.*

Criterion 1 decreases the number of choices to $\binom{32}{5}$.
Next, we analyze the property of $SD_i$. We expect that many active bits are obtained by applying $SD_i$, but there are efficient differential trails as summarized in Fig. 4 and we cannot avoid these trails[1]. As S-boxes diffuse each active bit to multiple rows, each $SD_i$ should not have the common efficient trails.

**Criterion 2.** *For any* $(i,j)$ *with* $i \neq j$ *and any* $(\alpha,\beta) \in \{1,2,3,6\} \times \{1,2,3,6\}$ *except for* $(2,6)$ *and* $(6,2)$*,*

$$\alpha \times r_i \neq (\beta \times r_j \bmod 64),$$
$$\alpha \times r_i \neq 64 - (\beta \times r_j \bmod 64).$$

Criterion 2 prevents two efficient trails from being connected via active S-boxes. Hopefully, while $(\alpha,\beta) = (2,6)$ and $(\alpha,\beta) = (6,2)$ should be included, it is not included because the number of candidates becomes nothing. Criterion 2 decreases the number of choices to 8704.
There are still many choices. Therefore, we next focus on the inverse of $SD_i$. Let us consider a Type II trail. We expect that there are many active S-boxes in one round before this efficient trail. Therefore, we apply $SD_j^{-1}$ to a 64-bit value whose Hamming weight is 2 and their distance is $r_i$ and guarantee many active bits.

**Criterion 3.** *For any* $i$ *and* $j$*, the Hamming weight of* $SD_j^{-1}(1 \oplus (1 \lll r_i))$ *is at least 10.*

Note that the value 10 is the maximum possible value among 8704 choices, and Criterion 3 decreases the number of choices to 512.
Hereinafter, we consider how to assign each rotation number to each row, i.e., we take the property of the Sycon S-box into consideration, and the number of candidates for $SD$ is now $512 \times 5!$. Recall Criterion 2, where we cannot include $(\alpha,\beta) = (6,2)$ and $(\alpha,\beta) = (2,6)$. To avoid efficient trails exploiting this property, we use the property of S-box. In both DDT and LAT, there is no possible propagation when input and output active bits are chosen from the same subspace from the following,

$$V(e_1,e_4) \qquad V(e_2,e_3) \qquad V(e_4,e_5),$$

where $e_i = (e_{i1},e_{i2},e_{i3},e_{i4},e_{i5}) \in \{0,1\}^5$ such that $e_{ij} = 1$ if and only if $i = j$. Here $V(e_i,e_j)$ means the vector space with basis $\{e_i,e_j\}$. One can note that $V(e_1,e_4) = \{00000, 10000, 00010, 10010\}$, and there is no possible transition from any value of $V(e_1,e_4)$ to any value of $V(e_1,e_4)$.

**Criterion 4.** *Only if* $(i,j) \in \{(1,4),(4,1),(2,3),(3,2),(4,5),(5,4)\}$*,*

$$2 \times r_i = (6 \times r_j \bmod 64),$$
$$2 \times r_i = 64 - (6 \times r_j \bmod 64)$$

*are possible.*

Criterion 4 guarantees that exploiting $2r_i = 6r_j \bmod 64$ (resp. $2r_i = 64 - 6r_j \bmod 64$) forces at least one row of other three rows to be active, and we can expect enough active bits are generated by additional active rows.
We further extend Criterion 3 to exploit the property of the S-box.

**Criterion 5.** *For any* $i$*,* $j_1$ *and* $j_2$ *such that there is* $x$ *satisfying*

$$\Pr[S(x) \oplus S(x \oplus (1 \lll j_1) \oplus (1 \lll j_2)) = (1 \lll i)] \neq 0,$$
$$\Pr[\langle x, (1 \ll i)\rangle = \langle S(x), (1 \ll j_1) \oplus (1 \ll j_2)\rangle] \neq 0,$$

$\max(wt(SD_{j_1}^{-1}(1 \oplus (1 \lll r_i))), wt(SD_{j_1}^{-1}(1 \oplus (1 \lll r_i))))$ *is at least 26.*

When we have a Type II trails with a single active row, Criterion 5 guarantees at least 26 active S-boxes one round before. After we applied these all criteria, 2048 candidates are left. Concering all of these candidates, we next evaluate $v_i$ and decide the good combination of $(r_4,r_3,r_2,r_1,r_0)$ and $(v_4,v_3,v_2,v_1,v_0)$.

[1] We have almost the same efficient linear trails, where the input and output of $SD_i$ are replaced.

**Choice of $v_i$.** The additional byte-wise rotation is introduced to increase the number of active S-boxes, and our goal is to guarantee 16 active S-boxes in both 3-round differential and linear trails. To choose each rotation number, we focus on the following four specific differential trails:

$$\xrightarrow{SB} 1 \xrightarrow{SD} 3 \xrightarrow{SB} 3 \xrightarrow{SD} \mathbf{X} \xrightarrow{SB},$$
$$\xrightarrow{SB} 2 \xrightarrow{SD} 2 \xrightarrow{SB} 2 \xrightarrow{SD} \mathbf{Y} \xrightarrow{SB},$$
$$\xrightarrow{SB} 4 \xrightarrow{SD} 2 \xrightarrow{SB} 2 \xrightarrow{SD} \mathbf{Z} \xrightarrow{SB},$$
$$\xrightarrow{SB} \mathbf{W} \xrightarrow{SD} 4 \xrightarrow{SB} 4 \xrightarrow{SD} 4 \xrightarrow{SB},$$

where each number shows the number of active columns in each state. Rotation numbers $v_i$'s are chosen such that $\mathbf{X}$, $\mathbf{Y}$, $\mathbf{Z}$, and $\mathbf{W}$ are maximized, i.e., $\mathbf{X} \geq 12$, $\mathbf{Y} \geq 12$, $\mathbf{Z} \geq 10$, and $\mathbf{W} \geq 8$ are necessary to guarantee 16 active S-boxes. We simultaneously focus on similar four specific linear trails and choose rotation numbers such that they also have enough active S-boxes.
We try out all byte rotations, where the rotation number for the last row is fixed to 0 and there are no common rotation numbers. As a result, we have 1088 candidates. Finally, our parameter uses one of these candidates, which are given by $(r_4, r_3, r_2, r_1, r_0) = (13, 21, 33, 55, 59)$ and $(v_4, v_3, v_2, v_1, v_0) = (0, 56, 16, 32, 40)$. Note that we are not sure that our design criteria is sufficient condition for 16 active S-boxes in 3 rounds. Therefore, we evaluated the lower bound of the number of active S-boxes by using a MILP. As a result, using our parameter guarantees 16 active S-boxes in 3-round for both differential and linear trails.

## 4.3 Choice of the AddRoundConst Layer

The purpose of the round constants is to break the symmetry in the round function of the permutation. As the maximum number of rounds is 12, we choose a linear feedback shift register (LFSR) with a primitive polynomial of degree 4 to generate the round constant values. Thus the hardware cost for this is minimal.

## 4.4 Choice of the Sycon Modes

Our choice is a *hardware efficient* mode for the Sycon permutation. We leverage the existing provably secure modes to instantiate the Sycon permutation to obtain two AEAD with different rates and one hash function. There are several lightweight variants of the sponge construction that are targeted to make the modes efficient and secure while keeping several other factors in mind such as efficient key absorption, proper domain separation for preventing attacks, and reducing the number of rounds of the permutation for efficiency. The mode of the Sycon AEAD algorithms is similar to SpongeWrap [14] and MonkeyDuplex [13], where the key absorption is inspired from the sLiSCP lightweight mode [4], and the domain separation is inspired from NORX [5]. Note that the key absorption at the rate part that happens before the absorption of AD gives additional security. For instance, if the attacker is able to get an internal state, then while inverting that to reach the initial state where key, nonce and IV were loaded, she has to exhaust the two 64-bit subkeys, meaning to exhaust $2^{128}$ option of possible keys.
For the hashing, we use the sponge mode of operation [15]. These modes are widely used and have been proven secure.

**Choice of Initial Vectors.** We constructed two AEAD instances and one hash instance using the Sycon permutation. An initial vector is used to uniquely identify an AEAD or hash instance of Sycon. The initial vectors "iv0" used in Sycon-AEAD-64, "iv1" used in Sycon-AEAD-96 and "iv2" used in Sycon-Hash-64 are obtained by taking the output of the Riemann Zeta function [23] evaluated at 2 and 3, respectively, and then taking the 19 decimal places in the hex representation to construct 64-bit initial vectors.

# 5 Security Analysis

In this section, we analyze the security of the Sycon permutation against cryptanalytic attacks such as differential and linear cryptanalysis, integral attack, cube attack, and differential-linear attack.

**Differential and Linear Cryptanalyses.** We estimated the minimum number of differentially and linearly active S-boxes by using MILP[37, 3]. Both numbers of differentially and linearly active S-boxes are 16 in 3 rounds. However, it is practically difficult to show the exact minimum number of active S-boxes in 4 rounds. Instead, we guarantee that the lower bound of the minimum number of active S-boxes in 4 rounds. We first generate the MILP model for 4 rounds and restrict the position of the active S-boxes in the 1st round. Then, we guarantee the number of active S-boxes in the last 3 rounds. By using this approach, we successfully guarantee that there are no differential and linear trails whose number of active S-boxes is lower than 24. Again, the number 24 is the lower bound of the minimum number of active S-boxes in 4 rounds. Therefore, we expect that the tight minimum number of active S-boxes is much higher. Actually, our heuristic search shows the existence of 4-round differential and linear trails with 42 and 43 active S-boxes, respectively.
Since there are 24 active S-boxes in 4 rounds, there are $64\,(= 24 + 24 + 16)$ and $72\,(= 24 + 24 + 24)$ active S-boxes in 11 and 12 rounds, respectively.

Table 6: Minimum number of active S-boxes

| # rounds | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Differential | 1 | 4 | 16 | $24 \leq, \leq 42$ |
| Linear | 1 | 4 | 16 | $24 \leq, \leq 43$ |

**Higher-order Differential, Integral, and Cube Attacks.** These attack methods exploit the low degree of the S-box, and the division property [38] is the most promising and powerful tool to evaluate the security against these attack methods. We evaluate whether the initialization is secure or not under the chosen nonce attack. Due to the S-box with degree 2, the trivial upper bound of the degree is $2^r$ in $r$ rounds. Seven rounds are not enough and the sum is independent of the secret key because $2^7 = 128$. Therefore, we evaluated the propagation of the division property for 8 rounds when all nonce bits are active. As a result, there is a division trail to any output, and it implies that the division property cannot find an 8-round integral distinguisher.
Cube attack is a kind of extension of the higher-order differential attack. The number of active bits is smaller than 128 and the goal is to recover the polynomial that is obtained by the sum of the output. The degree evaluation above already shows 8 rounds might involve terms whose degree is at least 128. Due to the substantial security margin, the cube attack is unlikely applied to Sycon.

**Differential-Linear Attacks.** Since the differential-linear attack is often applicable to ASCON, we also evaluated the security against the differential-linear attack. Like [21], we evaluated the experimental differential-linear distinguisher from 1-bit differential to 1-bit linear mask. As a result, there is a 4-round distinguisher whose correlation is $2^{-2.8}$, but we cannot observe a significant bias for 5 rounds experimentally. This result is almost the same as ASCON, and Sycon is as secure as ASCON against the differential-linear attack.

**Impossible Differential Attacks.** We searched for an impossible differential from 1 S-box active value to 1 S-box active value by using the MILP-based tool [35]. As a result, we found 5-round impossible differentials of the permutation, but we cannot find 6-round impossible differentials. As a similar observation is made by the

ASCON designers in [22], we have not found any practical attack using this property of the permutation.

# 6 Implementation Results

In this section, we present extensive hardware and software implementation results of Sycon, along with a detailed comparison with ASCON.

## 6.1 Hardware Implementation

We have implemented all variants of Sycon on ASIC and FPGA platforms along with the variants of ASCON for a fair comparison while optimizing the designs with respect to area. The hardware implementations were carried out using Verilog HDL. For FPGA, the design was synthesized using Xilinx Vivado while the ASIC synthesis was done using Mentor LeonardoSpectrum with 65nm logic process. The following are the details of the tool-chain:

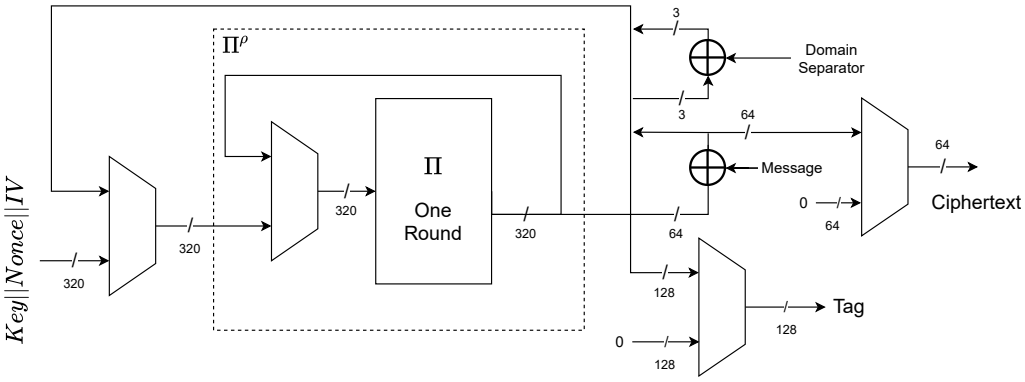
- **ASIC Synthesis**: Mentor LeonardoSpectrum Level 3 (2018a.2)
- **FPGA Synthesis**: Xilinx Vivado v2018.3 (64-bit)
- **Simulation**: ModelSim/Vivado Simulator.
- **FPGA platform**: Xilinx Kintex-7 (xc7k70tfbv676-2)
- **ASIC Cell library**: UMC 65 nm Low-Power RVT (Regular VT) Standard Performance Generic Core Cell Library from Faraday

The implementation of Sycon in both AEAD and hash modes is based on the standard round based iterative architecture. Entire state is processed using 64 parallel S-boxes. The round function is realized combinatorially. We use a 320-bit register to store the state after every iteration of the round function. For the authenticated encryption mode, after one application of the permutation, the message is absorbed into the state and domain separators are applied before the message is fed back. The ciphertext blocks and the tag are output as per the specification. The hashing mode is similar except the fact that the message is now XORed only in the absorption phase while the hash output happens in the squeezing phase as per the sponge construction. We furnish the datapaths of both the modes in Figure 5.
The FPGA and ASIC implementation results are summarized in Table 1. The component-wise details of ASIC area requirements for Sycon-AEAD-64 and Sycon-AEAD-96 are furnished in Figure 6. It is worth mentioning that the difference in area between the two variants primarily arises from the *Message Absorption* and *Output Generation* modules which is expected since it is directly correlated with the increase in the rate of absorption.

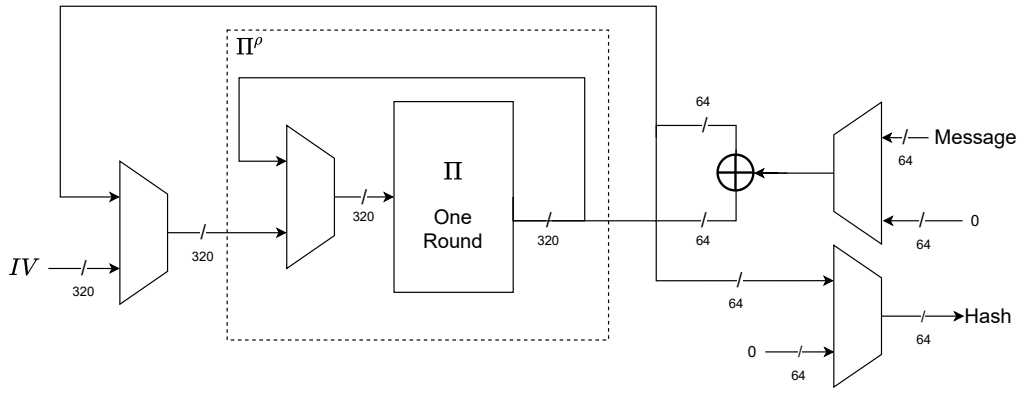
## 6.2 Hardware Optimization Rationale: Sycon vs ASCON

Our primary motivation while designing Sycon is to look at ASCON and find a scope for possible improvements. The point of investigation concentrated on the design of the internal permutation. Our research has led to optimized area footprint while achieving a higher maximum-frequency ($F_{Max}$), staying within an ASCON-type framework (see Table 1). One should compare the variants according to their exact or comparable block-sizes.

**Round Function.** The S-box selected for Sycon has an area footprint of 27.34 GE as compared to 28.12 GE for ASCON S-box in UMC 65nm ASIC technology. The savings in GE are amplified when we look at the entire substitution layer of Sycon consisting of 64 S-boxes for the 320-bit internal state of the permutation. The second and the most important optimization in Sycon is the choice of the inner rotation constants which introduces a high number of common subexpressions in the




(a) Sycon-AEAD-64 datapath for authenticated encryption mode.




(b) Sycon-Hash-64 datapath for hashing mode.

Figure 5: Datapath for AEAD and hash modes of Sycon.

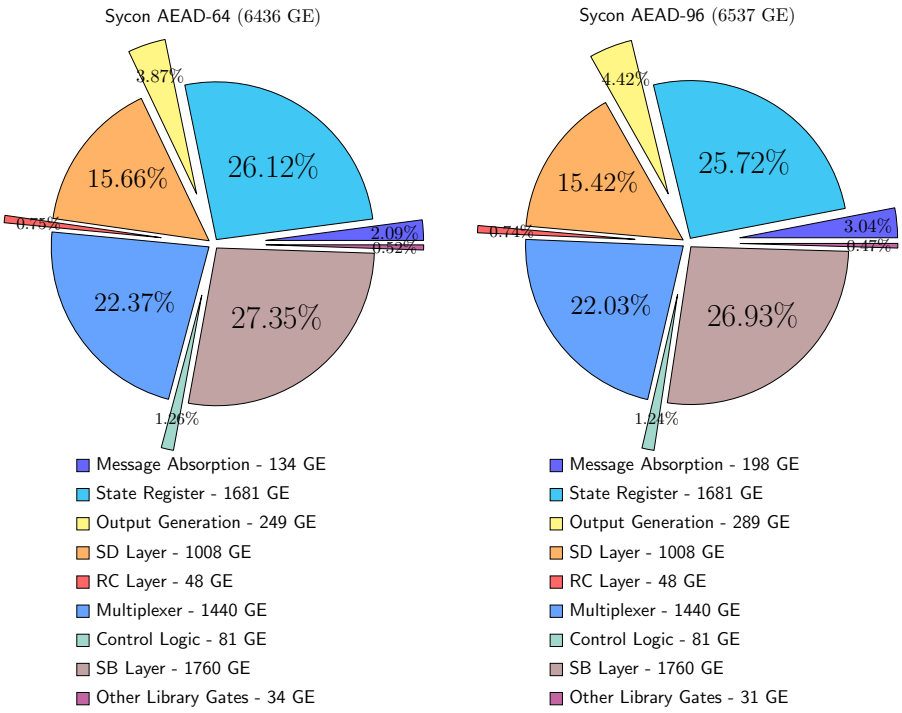



Figure 6: Component-wise ASIC area footprint for Sycon-AEAD-64 and Sycon-AEAD-96 in UMC 65nm technology

SubBlockDiffusion (SD) layer. This implies that many 3-input XOR gates have two inputs which are common and hence can be reused while computing 3-input XOR gates. The gain due to the resource sharing in the form of reusable 2-input XORs is immediately reflected in the FPGA and ASIC implementations of ASCON and Sycon round functions as captured in Table 7 and Figure 7, respectively. One can observe the absence of 3-input 64 bit XORs in Sycon as opposed to ASCON. This is due to the increase in 2-input 1-bit XORs owing to common subexpressions as stated above. In terms of ASIC design, one round of Sycon is 300 GE lighter than that of ASCON as depicted in Figure 7. The maximum area (around 89%) is saved in the SD layer which is a direct benefit of the subexpression elimination. The lighter S-box contributes to around 10% of the overall savings.

Table 7: RTL XOR component statistics comparison for one round of ASCON and Sycon permutation on Xilinx Kintex−7 (xc7k70tfbv676-2) platform

| **XOR Type** | | ASCON | Sycon |
|---|---|---|---|
| 2-input | 64-bit | 1 | 1 |
| 3-input | 64-bit | 5 | 0 |
| 2-input | 1-bit | 448 | 920 |
| 3-input | 1-bit | 128 | 112 |
| **Slice LUT** | | 486 | **410** |

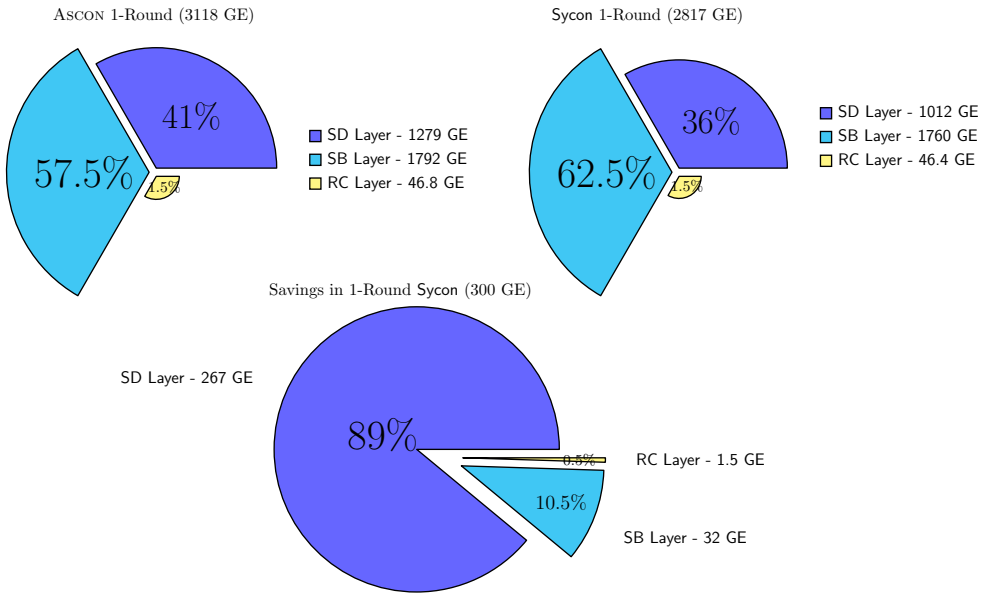



Figure 7: Comparative analysis of ASIC area savings of one round of Sycon and ASCON permutations.

**Mode of Operation.** Our idea behind the design optimization in the mode of operation is to avoid adding the key to the capacity part of the internal state.

As the key is absorbed through the rate part of the internal state, this saves XOR gates to the tune of the key-size thereby leading to significant gains for Sycon in terms of area. The same is reflected in Table 1 where we observe the major difference in the LUT count between Sycon-AEAD-64 and ASCON-128. For a fair comparison, we carried out the implementations with as much similarity as possible across instances and ciphers.

For the purpose of this analysis we perform a rough segregation of the ASIC design modules. We take into account the part of the design exclusively employed to implement the mode. This includes the handling of the message absorption phase. It can be noted that this is the place where Sycon differs sharply from ASCON which is attributed to the fact that Sycon absorbs the second whitening key through the rate part as opposed to ASCON which uses the capacity part. This leads to $\left\lceil \frac{\text{Key-Size}}{\text{Block-Size}} \right\rceil$ extra permutation calls but saves real-estate in terms of valuable area footprint. The analysis is captured in the form of Figure 8 which shows a massive gain of 832 GE in Sycon mode. Finally, as can be seen from Table 1, the overall area footprint for Sycon-AEAD-64 stands at 6436 GE as compared to 7567 GE for ASCON-128 which translates to substantial savings of 14.95% in area.

Another interesting observation from Table 1 is the difference between Sycon and ASCON as regards the maximum frequency reported for ASIC. We did a critical path analysis for the same. In case of ASCON the critical path consists of the message absorption module finally leading up to the 320-bit buffer that is being used in the design. On the contrary, the critical path for Sycon is between the round-counter module (which houses the control logic for keeping track of the current round and also generates the `permutation-done` signal) and the 320-bit buffer. The inclusion of the message absorption module in the critical path of ASCON showcases the additional delay it incurs due to adding the key in the capacity part. One can note that in case of ASCON, the key is added to *various parts* of the capacity part based on the domain of operation. This translates to additional MUX-es in hardware which are supposed

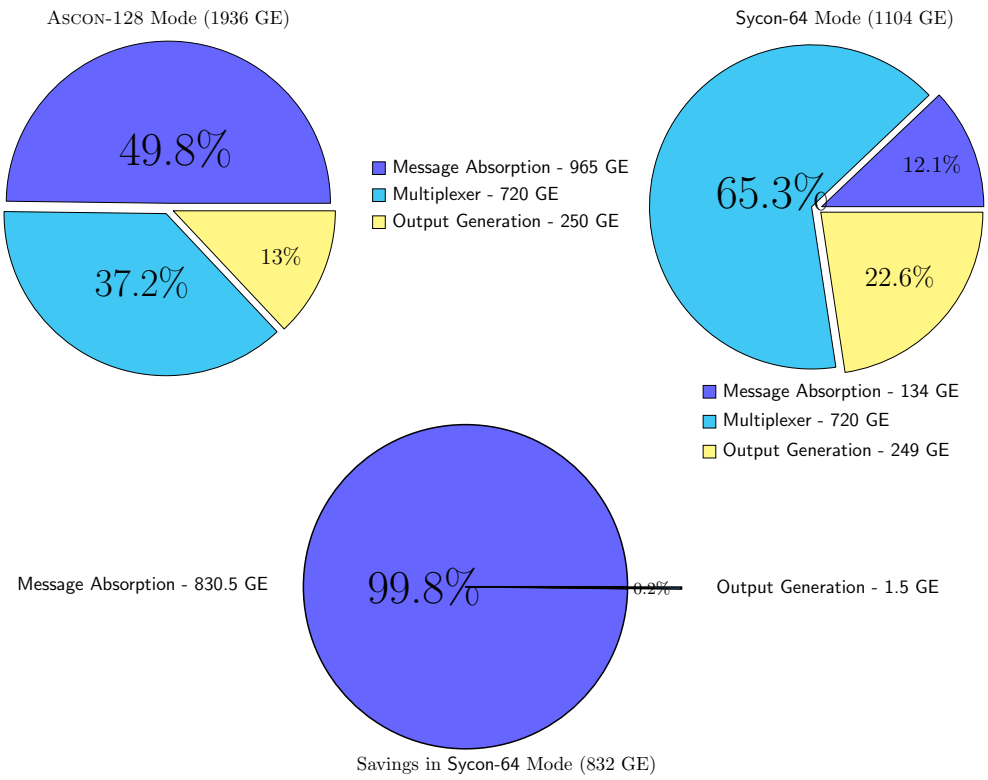



Figure 8: Comparative analysis of ASIC area savings of the mode of operation of Sycon permutation with respect to ASCON.

to contribute to the delay. In contrast, the mode chosen by Sycon only requires the domain separators to be added thereby saving additional gates (as prevalent from the area savings) and signal delay. To appreciate this further note that ASCON and Sycon hashing modes have same $F_{Max}$ since the capacity part no longer differs due to the same mode being used. To put things into perspective Sycon-64 clocks a frequency of 1.826 GHz as against 1.242 GHz by ASCON-128 thereby registering an huge improvement of 42.8%.

## 6.3 Hardware Performance Analysis

**Latency, Throughput, Power and Energy Measurement.** We adopt a different philosophy while measuring the latency of our design. This is motivated by the fact that a single block latency for authenticated ciphers that have both initializations and finalization does not seem to paint the right picture. For instance, in case of ASCON, a single block computation entails an additional overhead of two permutation calls (one each for initialization and finalization) resulting in an additional 24 clock-cycles. For Sycon the same figure is 60. However, if we use longer messages and then scale it back to a single block, then the additional overhead is amortized and we get a more comparable setting. In line with this, we calculate the *amortized* latency which is a function of the message-size to calculate other dependent parameters like throughput and energy. As a rule of thumb for lightweight designs, we calculate the results for the conventional 100 KHz frequency and repeat the same for the maximum frequency of a variant reported by the synthesis tool. For power, we rely on the standard power of 5.68 ($nW/MHz/GE$) as per the data available for 65nm CMOS technology node [25].

**Comparative Performance Metrics.** We give a detailed comparative analysis across various standard parameters in terms of hardware performance which is captured in Tables 8 and 9. It is worth noting that latencies depicted in the tables are actually the *amortized latencies* determined by the corresponding message-sizes as stated earlier. It can be observed that as the message-size increases the latencies of ASCON and Sycon variants of comparable block-sizes start to converge. This is due to the fact that the effect of the extra overhead in terms of additional permutation calls that Sycon embraces to avoid adding the key to the capacity part (to save XOR gates) slowly diminishes as the message size increases. This in turn leads to the visible increase in hardware efficiency at higher message lengths as seen in both Tables 8 and 9. For instance, for a message-size of 256 bits at 100 KHz Sycon-AEAD-64 has an efficiency of 47.35 Kbps/KGE as compared to 70.48 Kbps/KGE for ASCON-128. On the other hand, when the message-size increases to 4096 bits, the efficiency of Sycon-AEAD-64 increases to 143.34 Kbps/KGE surpassing ASCON-128 which is at 132.67 Kbps/KGE.

*Remark* 1. From the above, it is evident that Sycon performs better than ASCON for reasonable and quite practical message-sizes, which is attributed to about 48% higher maximum-frequency and 14.95% lower area that Sycon is able to achieve.

Table 8: Performance metrics analysis of of Sycon and ASCON at UMC 65nm technology at standard frequency of 100 KHz for lightweight constructions

| Primitive | Message Size (bits) | Frequency (KHz) | Block-Size | Latency (Cycles/block) | Throughput (Kbps) | Area (GE) | Efficiency (Kbps/KGE) | Power ($\mu$W) | Energy ($\mu$J) |
|---|---|---|---|---|---|---|---|---|---|
| Sycon-AEAD-64 | 256 | 100 | 64 | 21.00 | 304.76 | 6436 | 47.35 | 3.66 | 1.20 |
| Sycon-AEAD-96 | | | 96 | 27.00 | 355.56 | 6537 | 54.39 | 3.71 | 1.04 |
| Sycon-Hash-64 | | | 64 | 24.00 | 266.67 | 6038 | 44.16 | 3.43 | 1.29 |
| ASCON-128 | | | 64 | 12.00 | 533.33 | 7567 | 70.48 | 4.30 | 0.81 |
| ASCON-128a | | | 128 | 20.00 | 640.00 | 7614 | 84.06 | 4.32 | 0.68 |
| ASCON-Hash | | | 64 | 24.00 | 266.67 | 6319 | 42.20 | 3.59 | 1.35 |
| Sycon-AEAD-64 | 1024 | | 64 | 9.75 | 656.41 | 6436 | 101.99 | 3.66 | 0.56 |
| Sycon-AEAD-96 | | | 96 | 12.45 | 770.80 | 6537 | 117.91 | 3.71 | 0.48 |
| Sycon-Hash-64 | | | 64 | 15.00 | 426.67 | 6038 | 70.66 | 3.43 | 0.80 |
| ASCON-128 | | | 64 | 7.50 | 853.33 | 7567 | 112.77 | 4.30 | 0.50 |
| ASCON-128a | | | 128 | 11.00 | 1163.64 | 7614 | 152.83 | 4.32 | 0.37 |
| ASCON-Hash | | | 64 | 15.00 | 426.67 | 6319 | 67.52 | 3.59 | 0.84 |
| Sycon-AEAD-64 | 4096 | | 64 | 6.94 | 922.52 | 6436 | 143.34 | 3.66 | 0.40 |
| Sycon-AEAD-96 | | | 96 | 8.40 | 1143.49 | 6537 | 174.93 | 3.71 | 0.32 |
| Sycon-Hash-64 | | | 64 | 12.75 | 501.96 | 6038 | 83.13 | 3.43 | 0.68 |
| ASCON-128 | | | 64 | 6.38 | 1003.92 | 7567 | 132.67 | 4.30 | 0.43 |
| ASCON-128a | | | 128 | 8.75 | 1462.86 | 7614 | 192.13 | 4.32 | 0.30 |
| ASCON-Hash | | | 64 | 12.75 | 501.96 | 6319 | 79.44 | 3.59 | 0.72 |

## 6.4 Efficiency in Software

**Bit-slice Efficiency on 64-bit CPUs.** We report the speed of the Sycon permutation, AEAD and hash algorithms and compare it with other permutations. We have implemented the Sycon permutation, AEAD and hash algorithms in the bit-sliced fashion using the SIMD Intel Intrinsics including SSE2 and AVX2. The SSE2 supports operations on 128-bit XMM registers and AVX2 supports operations on 256-bits YMM registers. The codes were compiled using the `gcc 5.4.0` compiler with `-g -Wall -O2 -fomit-frame-pointer -funroll- all-loops` flags (Skylake, i7-6700 CPU). The implementation details are given in Appendix B.

**Benchmarking.** We report the speed of the Sycon permutation, authenticated encryption and hash algorithms for both SSE2 and AVX2 implementations. The speed is measured in terms of the number of clock cycles per byte (c/B). Table 10

Table 9: Performance metrics analysis of of Sycon and ASCON at UMC 65nm technology at maximum frequency ($F_{Max}$) reported by synthesis tool for each variant

| Primitive | Message Size (bits) | Frequency (GHz) | Block-Size | Latency (Cycles/block) | Throughput (Gbps) | Area (GE) | Efficiency (Gbps/KGE) | Power ($m$W) | Energy ($m$J) |
|---|---|---|---|---|---|---|---|---|---|
| Sycon-AEAD-64 | 256 | 1.826 | 64 | 21.00 | 5.56 | 6436 | 0.86 | 66.75 | 21.90 |
| Sycon-AEAD-96 | | 1.826 | 96 | 27.00 | 6.49 | 6537 | 0.99 | 67.80 | 19.07 |
| Sycon-Hash-64 | | 1.884 | 64 | 24.00 | 5.02 | 6038 | 0.83 | 64.61 | 24.23 |
| ASCON-128 | | 1.242 | 64 | 12.00 | 6.62 | 7567 | 0.88 | 53.38 | 10.01 |
| ASCON-128a | | 1.246 | 128 | 18.00 | 8.86 | 7614 | 1.16 | 53.89 | 7.58 |
| ASCON-Hash | | 1.884 | 64 | 24.00 | 5.02 | 6319 | 0.80 | 67.62 | 25.36 |
| Sycon-AEAD-64 | 1024 | 1.826 | 64 | 9.75 | 11.99 | 6436 | 1.86 | 66.75 | 10.17 |
| Sycon-AEAD-96 | | 1.826 | 96 | 12.45 | 14.07 | 6537 | 2.15 | 67.80 | 8.80 |
| Sycon-Hash-64 | | 1.884 | 64 | 15.00 | 8.04 | 6038 | 1.33 | 64.61 | 15.14 |
| ASCON-128 | | 1.242 | 64 | 7.50 | 10.60 | 7567 | 1.40 | 53.38 | 6.26 |
| ASCON-128a | | 1.246 | 128 | 9.00 | 17.72 | 7614 | 2.33 | 53.89 | 3.79 |
| ASCON-Hash | | 1.884 | 64 | 15.00 | 8.04 | 6319 | 1.27 | 67.62 | 15.85 |
| Sycon-AEAD-64 | 4096 | 1.826 | 64 | 6.94 | 16.85 | 6436 | 2.62 | 66.75 | 7.24 |
| Sycon-AEAD-96 | | 1.826 | 96 | 8.40 | 20.88 | 6537 | 3.19 | 67.80 | 5.93 |
| Sycon-Hash-64 | | 1.884 | 64 | 12.75 | 9.46 | 6038 | 1.57 | 64.61 | 12.87 |
| ASCON-128 | | 1.242 | 64 | 6.38 | 12.47 | 7567 | 1.65 | 53.38 | 5.32 |
| ASCON-128a | | 1.246 | 128 | 6.75 | 23.63 | 7614 | 3.10 | 53.89 | 2.84 |
| ASCON-Hash | | 1.884 | 64 | 12.75 | 9.46 | 6319 | 1.50 | 67.62 | 13.47 |

presents the speed for the Sycon permutation. The best speed achieved by the Sycon permutation is 2.20 c/B in the AVX2 implementation. We also implement the ASCON permutation using SSE2 and AVX2 and compared our result with it. As our diffusion layer has four extra left rotations, the speed of the Sycon permutation is slightly decreased compared to ASCON. We use Gimli and ACE codes and run on the same machine for a fair comparison. When measuring the speed of Sycon-AEAD-64 with AVX2 implementations, we choose plaintext messages of lengths 64 and 1536 bytes and an associated data of length 128 bits where the speed computation includes all four phases. Table 11 presents the speed of two AEAD instances and the hash instance. In Appendix B, Table 13 presents the software efficiency on three different microcontrollers.

Table 10: A speed-comparison of the Sycon permutation with others on 64-bit Skylake CPUs. The speed is measured in terms of clock cycles per byte (c/B).

| **Permutation** | **Speed (c/B)** | | **Ref** |
|---|---|---|---|
| | AVX2 | SSE2 | |
| Sycon (four instances) | 2.20 | 4.22 | **this paper** |
| ASCON (four instances) | 1.98 | 3.72 | **this paper**, [22] |
| Gimli (four instances) | 1.57 | 4.18 | [12] |
| ACE (eight instances) | 9.97 | 15.66 | [2] |

Table 11: The speed of the AEAD and hash algorithms on 64-bit Skylake CPUs.

| | **Message length** | | | |
|---|---|---|---|---|
| | **64 bytes** | | **1536 bytes** | |
| | **Speed (c/B)** | | **Speed (c/B)** | |
| **Algorithms** | AVX2 | SSE2 | AVX2 | SSE2 |
| Sycon-AEAD-64 | 20.19 | 30.72 | 7.33 | 11.28 |
| Sycon-AEAD-96 | 13.02 | 22.45 | 6.26 | 9.57 |
| Sycon-Hash-64 | 25.42 | 41.19 | 10.96 | 22.50 |

# 7 Conclusions

In this paper, we have proposed Sycon which is the lightest member in the ASCON family. We have analyzed the Sycon permutation against the powerful distinguishing attacks and showed that it withstands those attacks well. Our performance comparisons demonstrate that the Sycon is much lighter in hardware than ASCON while at the same time receives a better maximum frequency. So this puts Sycon ahead of ASCON for real world applications.

# References


[1] CAESAR: Competition for authenticated encryption: Security, applicability, and robustness (2012), https://competitions.cr.yp.to/caesar.html

[2] Aagaard, M., AlTawy, R., Gong, G., Mandal, K., Rohit, R.: ACE: An authenticated encryption and hash algorithm. NIST Lightweight Cryptography Round 2 (2019)

[3] Abdelkhalek, A., Sasaki, Y., Todo, Y., Tolba, M., Youssef, A.M.: MILP modeling for (large) s-boxes to optimize probability of differential characteristics. IACR Trans. Symmetric Cryptol. **2017**(4), 99–129 (2017). https://doi.org/10.13154/tosc.v2017.i4.99-129, https://doi.org/10.13154/tosc.v2017.i4.99-129

[4] AlTawy, R., Rohit, R., He, M., Mandal, K., Yang, G., Gong, G.: sLiSCP: Simeck-based permutations for lightweight Sponge cryptographic primitives. In: Adams, C., Camenisch, J. (eds.) Selected Areas in Cryptography – SAC 2017. pp. 129–150. Springer International Publishing, Cham (2018)

[5] Aumasson, J.P., Jovanovic, P., Neves, S.: Norx: parallel and scalable AEAD. In: European Symposium on Research in Computer Security. pp. 19–36. Springer (2014)

[6] Banik, S., Bogdanov, A., Isobe, T., Shibutani, K., Hiwatari, H., Akishita, T., Regazzoni, F.: Midori: A block cipher for low energy. In: Iwata, T., Cheon, J.H. (eds.) Advances in Cryptology - ASIACRYPT, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9453, pp. 411–436. Springer (2015)

[7] Banik, S., Bogdanov, A., Luykx, A., Tischhauser, E.: SUNDAE: small universal deterministic authenticated encryption for the internet of things. IACR Trans. Symmetric Cryptol. **2018**(3), 1–35 (2018)

[8] Banik, S., Bogdanov, A., Peyrin, T., Sasaki, Y., Sim, S.M., Tischhauser, E., Todo, Y.: SUNDAE-GIFT. NIST Lightweight Cryptography Round 2 (2019)

[9] Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: GIFT: A small present - towards reaching the limit of lightweight encryption. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES. Lecture Notes in Computer Science, vol. 10529, pp. 321–345. Springer (2017)

[10] Bao, Z., Chakraborti, A., Datta, N., Guoa, J., Nandi, M., Peyrin, T., Yasuda, K.: PHOTON-BEETLE. NIST Lightweight Cryptography Round 2 (2019)

[11] Beierle, C., Jean, J., Kölbl, S., Leander, G., Moradi, A., Peyrin, T., Sasaki, Y., Sasdrich, P., Sim, S.M.: The SKINNY family of block ciphers and its low-latency variant MANTIS. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9815, pp. 123–153. Springer (2016)

[12] Bernstein, D.J., Kölbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F.X., Todo, Y., et al.: Gimli: a cross-platform permutation. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 299–320. Springer (2017)

[13] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Permutation-based encryption, authentication and authenticated encryption. Presented at DIAC (2012)

[14] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Duplexing the Sponge: single-pass authenticated encryption and other applications. In: International Workshop on Selected Areas in Cryptography. pp. 320–337. Springer (2011)

[15] Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Keccak specifications (2009)

[16] Biham, E., Shamir, A.: Differential cryptanalysis of DES-like cryptosystems. In: Proceedings of the 10th Annual International Cryptology Conference on Advances in Cryptology. pp. 2–21. CRYPTO (1991)

[17] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: Paillier, P., Verbauwhede, I. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer (2007)

[18] Chakraborti, A., Datta, N., Nandi, M., Yasuda, K.: Beetle family of lightweight and secure authenticated encryption ciphers. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(2), 218–241 (2018)

[19] Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography, Springer (2002)

[20] Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Mennink, B., Primas, R., Unterluggauer, T.: ISAP v2.0. NIST Lightweight Cryptography Round 2 (2019)

[21] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Cryptanalysis of ASCON. In: Nyberg, K. (ed.) Topics in Cryptology - CT-RSA. Lecture Notes in Computer Science, vol. 9048, pp. 371–387. Springer (2015)

[22] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: ASCON v1.2. NIST Lightweight Cryptography Round 2 (2019), CAESAR, finalist

[23] Edwards, H.: Riemann's Zeta function. Dover Publications (2001)

[24] Guo, J., Peyrin, T., Poschmann, A.: The PHOTON family of lightweight hash functions. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 222–239. Springer (2011). https://doi.org/10.1007/978-3-642-22792-9_13, https://doi.org/10.1007/978-3-642-22792-9_13

[25] Hatzivasilis, G., Fysarakis, K., Papaefstathiou, I., Manifavas, C.: A review of lightweight block ciphers. J. Cryptographic Engineering **8**(2), 141–184 (2018)

[26] Inc., G.O.: Gurobi optimizer. Official webpage, http://www.gurobi.com/

[27] Matsui, M.: Linear cryptanalysis method for DES cipher. In: Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology. pp. 386–397. EUROCRYPT '93, Springer-Verlag New York, Inc. (1994)

[28] NIST: NIST Lightweight Cryptography project. https://csrc.nist.gov/Projects/Lightweight-Cryptography/ (2019)

[29] NIST: Round 1 of the NIST lightweight cryptography project (2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-1-Candidates

[30] NIST: Round 2 of the nist lightweight cryptography project (2019), https://csrc.nist.gov/Projects/Lightweight-Cryptography/Round-2-Candidates

[31] Penazzi, D., Montes, M.: SHAMASH. NIST Lightweight Cryptography Round 1 (2019)

[32] Riou, S.: DRYGASCON. NIST Lightweight Cryptography Round 2 (2019)

[33] Sarkar, S., Mandal, K., Saha, D.: On the relationship between resilient Boolean functions and linear branch number of S-boxes. In: Hao, F., Ruj, S., Gupta, S.S. (eds.) Progress in Cryptology - INDOCRYPT. Lecture Notes in Computer Science, vol. 11898, pp. 361–374. Springer (2019)

[34] Sarkar, S., Mandal, K., Saha, D.: SYCON v1.0. NIST Lightweight Cryptography Round 1 (2019)

[35] Sasaki, Y., Todo, Y.: New impossible differential search tool from design and cryptanalysis aspects - revealing structural properties of several ciphers. In: Coron, J., Nielsen, J.B. (eds.) Advances in Cryptology - EUROCRYPT, Proceedings, Part III. Lecture Notes in Computer Science, vol. 10212, pp. 185–215 (2017)

[36] Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Biryukov, A. (ed.) Fast Software Encryption FSE. Lecture Notes in Computer Science, vol. 4593, pp. 181–195. Springer (2007)

[37] Sun, S., Hu, L., Wang, P., Qiao, K., Ma, X., Song, L.: Automatic security evaluation and (related-key) differential characteristic search: Application to simon, present, lblock, DES(L) and other bit-oriented block ciphers. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I. Lecture Notes in Computer Science, vol. 8873, pp. 158–178. Springer (2014). https://doi.org/10.1007/978-3-662-45611-8_9, https://doi.org/10.1007/978-3-662-45611-8_9

[38] Todo, Y.: Structural evaluation by generalized integral property. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 287–314. Springer (2015)

# A Description of Sycon AEAD and Hash Modes

**Sycon AEAD mode.** The authenticated encryption algorithm accepts as input a secret key $K$ of length $\kappa$, a public nonce $N$ of length $n$, an (optional) associated data $A$, and a plaintext message $M$ and outputs a ciphertext $C$ of the same length as of the message and a tag $T$ of length $\tau$. Similarly, The decryption algorithm accepts as input a secret key $K$, a public nonce $N$, an (optional) associated data $A$, a ciphertext message $C$, and a tag $T$ and computes a tag $T'$. It outputs the plaintext message $M$ if the verification of the tag succeeds, otherwise, outputs $\perp$. A high-level overview of the encryption algorithm is provided in Fig. 2. The encryption/decryption process consists of four distinct phases, namely the initialization phase, processing associated data, encrypting the message and generation of the tag.

**Rate and capacity parts:** The internal state is divided into two parts, called rate part and capacity part. Any input that is absorbed into the state is done through the rate part. Given the internal state $\mathbf{s} = \mathbf{s}_0\|\mathbf{s}_1\|\mathbf{s}_2\|\mathbf{s}_3\|\mathbf{s}_4$, for $r = 64$, $\mathbf{s}_0$ serves as the rate part and the remaining part is the capacity part. For $r = 96$, the entire $\mathbf{s}_0$ and the first half of $\mathbf{s}_1$ serves as the rate part in the state, i.e., $\mathbf{s}_0\|\lfloor\mathbf{s}_1\rfloor_{32}$, and the remaining part of the state is the capacity.

**Padding:** When the length of the plaintext or associated data is not a multiple of $r$, padding is mandatory to make the message block multiple of $r$. For empty associated data, no padding is applied, otherwise, the padded associated data $A$ is $\text{PAD}_r(A) = A\|1\|0^{r-1-|A| \bmod r}$. For the plaintext message $M$, the padding is applied as $\text{PAD}_r(M) = M\|1\|0^{r-1-|M| \bmod r}$. For $\kappa = 128$ and $r = 64$, when the key is absorbed into the state, no padding is required. Whereas, for $r = 96$, the padding for the key is required, which is described as $\text{PAD}_r(K) = K\|1\|0^{r-1-|K| \bmod r} = K_0\|K_1$.

**Initialization:** The initialization phase consists of a loading phase that loads the key and nonce to the state and absorbs the key into the state. The key in $\mathbf{s}_0$ and $\mathbf{s}_1$, nonce in $\mathbf{s}_2$ and $\mathbf{s}_3$ and initial vector in $\mathbf{s}_4$ are loaded and then the full round Sycon permutation applied on the state, followed by absorbing the padded key through the rate part with two permutation calls.

**Processing associated data:** This algorithm is applied after the initialization phase if the associated data is nonempty. It accepts the associated data (AD) and the current state as input and returns the state of the permutation. The padding on associated data $A$ is applied as $\text{PAD}_r(A) = A_0 \| \cdots \| A_{n-1}$ where $n$ is the number of blocks. For each block of the padded associated data, it is XORed with the rate part of the current state, followed by applying the permutation. Note that there is no output during the AD processing.

**Encryption/decryption:** After processing the associated data, the encryption algorithm is applied on the plaintext $M$. First, the padding rule $(10^*)$ is applied on the plaintext $M$, and the padding on $M$ returns a padded message which is a multiple of $r$, i.e., $\text{PAD}_r(M) = M_0 \| \cdots \| M_{m-1}$, where $m$ is the number blocks for the padded message. The encryption algorithm produces a ciphertext of length $m$ corresponding to the input plaintext. For each message block $M_i$, the contents in the rate part serves as a keystream to which the message block $M_i$ is XORed, which is the ciphertext $C_i$. Then the Sycon permutation is applied to update the state to encrypt the next message block. The decryption process is the same, except the message is replaced by the ciphertext and the contents in the rate part is replaced by the ciphertext block.

**Finalization:** After the encryption or decryption algorithm, the finalization phase is applied to output a 128-bit tag. In this phase, the key is again absorbed to the state through the rate by two permutation calls. Given the state $\mathbf{s} = \mathbf{s}_0 \| \mathbf{s}_1 \| \mathbf{s}_2 \| \mathbf{s}_3 \| \mathbf{s}_4$, the tag extraction function, denoted by $\texttt{ExtTag}(\mathbf{s})$, extracts the tag from the state by concatenating contents from $\mathbf{s}_2$ and $\mathbf{s}_3$, i.e, $\texttt{ExtTag}(\mathbf{s}) = \mathbf{s}_2 \| \mathbf{s}_3$.

Table 12: The initial vectors (IVs) used in the Sycon modes

| IV ID | Initial vector | IV used in algorithm |
|---|---|---|
| iv0 | 0x0000000000000000 | Sycon-AEAD-64 |
| iv1 | 0x5980A92AFC5D9D2C | Sycon-AEAD-96 |
| iv2 | 0x1C0A80D42C6E63C5 | Sycon-Hash-64 |

**Sycon hash mode.** A message digest computation consists of the following two steps, namely absorbing/processing the message and squeezing/outputting the hash value or digest.

**Processing message:** To process a message $M$, the padding is applied, i.e., $\text{PAD}_r(M) = M_0 \| \cdots \| M_{m-1}$ where $m$ is the number blocks for the padded message. Before start processing the message, the predefined initial vector (iv2) is loaded into $\mathbf{s}_2$ of the state and the remaining state bits are set to zero, followed by applying the Sycon permutation. After that, sequentially each message block is XORed with the rate part and then the permutation is applied until the last message block.

**Outputting digest:** After absorbing the message in the state, it outputs a message digest of 256 bits by outputting four blocks of $r = 64$ bits from the rate part where the permutation is invoked after taking $r$-bits from the rate part.

# B Details on Efficiency in Software

**Round function.** We briefly explain how the Sycon permutation is implemented in AVX2. We use five YMM registers ($R_i$), pacing four instances of the permutation, where $\mathbf{s}^i = \mathbf{s}^i_4 \| \mathbf{s}^i_3 \| \mathbf{s}^i_2 \| \mathbf{s}^i_1 \| \mathbf{s}^i_0$ represents the state of the $i$-th instance, and register contents are:

$$R_0 = \mathbf{s}_0^3 \| \mathbf{s}_0^2 \| \mathbf{s}_0^1 \| \mathbf{s}_0^0;$$
$$R_1 = \mathbf{s}_1^3 \| \mathbf{s}_1^2 \| \mathbf{s}_1^1 \| \mathbf{s}_1^0;$$
$$R_2 = \mathbf{s}_2^3 \| \mathbf{s}_2^2 \| \mathbf{s}_2^1 \| \mathbf{s}_2^0;$$
$$R_3 = \mathbf{s}_3^3 \| \mathbf{s}_3^2 \| \mathbf{s}_3^1 \| \mathbf{s}_3^0;$$
$$R_4 = \mathbf{s}_4^3 \| \mathbf{s}_4^2 \| \mathbf{s}_4^1 \| \mathbf{s}_4^0;$$

The S-box is chosen so that the permutation has an efficient bit-slice implementation. The bit-slice form of the S-box is as follows.

$$\begin{array}{lllll}
\mathtt{t}_0 = \mathtt{x}_2 \,\hat{}\, \mathtt{x}_4 & \mathtt{t}_1 = \mathtt{t}_0 \,\hat{}\, \mathtt{x}_1 & \mathtt{t}_2 = \mathtt{x}_1 \,\hat{}\, \mathtt{x}_3 & \mathtt{t}_3 = \mathtt{x}_0 \,\hat{}\, \mathtt{x}_4 & \mathtt{t}_4 = \mathtt{t}_1 \& \mathtt{x}_3 \\
\mathtt{t}_5 = \mathtt{t}_3 \,\hat{}\, \mathtt{t}_4 & \mathtt{x}_1 = \mathtt{\sim x}_1 & \mathtt{x}_1 = \mathtt{x}_1 \& \mathtt{x}_3 & \mathtt{t}_6 = \mathtt{\sim t}_2 & \mathtt{t}_6 = \mathtt{t}_6 \& \mathtt{x}_0 \\
\mathtt{x}_1 = \mathtt{x}_1 \,\hat{}\, \mathtt{t}_1 & \mathtt{x}_1 = \mathtt{x}_1 \,\hat{}\, \mathtt{t}_6 & \mathtt{t}_3 = \mathtt{\sim t}_3 & \mathtt{t}_6 = \mathtt{t}_3 \& \mathtt{x}_2 & \mathtt{t}_1 = \mathtt{t}_6 \,\hat{}\, \mathtt{t}_2 \\
\mathtt{t}_0 = \mathtt{\sim t}_0 & \mathtt{x}_3 = \mathtt{t}_0 \& \mathtt{x}_3 & \mathtt{x}_3 = \mathtt{x}_3 \,\hat{}\, x_0 & \mathtt{x}_2 = \mathtt{\sim x}_2 & \mathtt{x}_3 = \mathtt{x}_3 \,\hat{}\, \mathtt{x}_2 \\
\mathtt{x}_4 = \mathtt{\sim x}_4 & \mathtt{x}_4 = \mathtt{x}_4 \& \mathtt{x}_0 & \mathtt{x}_4 = \mathtt{x}_4 \,\hat{}\, \mathtt{t}_2 & \mathtt{x}_0 = \mathtt{t}_5 & \mathtt{x}_2 = \mathtt{t}_1
\end{array}$$

We need 22 logical instructions including not operations to implement the Sbox, which is the same as ASCON's S-box. For the `SB` layer, the instructions such as `vpxor`, `vpand` and `vpandn` among the registers are used according to the bit-slice representation of the S-box to implement this layer. On the other hand, for the `SD` layer, the instructions `vpsllq` and `vpsrlq` are used to implement it, and there are no cross-register operations required.

**Efficiency on microcontroller.** To assess the software performance of Sycon on microcontrollers, we have implemented the Sycon authenticated encryption and hash algorithms on the 8-bit Atmel Atmega32 and a 32-bit MIPS32 from MIPS Technologies. The 8-bit Atmel Atmega32 microcontroller has 2 Kbytes of flash, 32 KBytes of RAM and 32 8-bit general purpose registers. MIPS32 has 32 32-bit general purpose registers. We implement the Sycon instance in assembly, and the AVR Simulator IDE was used to write the code. In our implementations, we implement the S-box in the bitsliced fashion, instead of a look-up table, to achieve highest efficiency while reducing memory. We use a plaintext of 72 bytes in our experiment to obtain cycles for AEAD and hash instances. For instance, the Sycon permutation evaluation requires 12,097 cycles on the MIPS32 microcontroller, and the throughput of the permutation is 302.43 cycles/byte. Table 13 presents the cycle counts, code sizes in bytes and cycles per byte for the Sycon permutation and the AEAD and hash algorithms.

Table 13: Performance of the Sycon permutation and its AEAD and hash modes on 8-bit Atmel Atmega32 and 32-bit MIPS32 microcontrollers

| | Sycon | | Sycon-AEAD-64 | | Sycon-AEAD-96 | | Sycon-Hash-64 | |
|---|---|---|---|---|---|---|---|---|
| Platform | 8-bit | 32-bit | 8-bit | 32-bit | 8-bit | 32-bit | 8-bit | 32-bit |
| Cycles | 20,009 | 12,097 | 217,254 | 129,113 | 202,969 | 122,349 | 260,915 | 159,362 |
| Code Size(Bytes) | 978 | 1,210 | 1,267 | 1,481 | 1,271 | 1,495 | 1,173 | 1,401 |
| Cycles/Byte | 500.23 | 302.43 | 3,017.29 | 1,793.24 | 2,819.01 | 1,699.29 | 3,623.82 | 2,213.36 |

# C Efficient Differential Trail of Sycon v1.0

The main reason why we tweaked the design from v1.0 is finding of an efficient 4-round differential trail. Figure 9 shows a 4-round differential trail with only 21 active S-boxes. Considering the best differential trail in [34] has 51 active S-boxes, this differential trail is dramatically efficient. This negative result of Sycon v1.0 comes from heuristic design of the diffusion layer. As a result, very efficient differential trails were overlooked. In our new design, we carefully analyzed the condition that efficient trails happen, and the diffusion layer is designed more systematically.

# D Test Vectors

We present the test vectors for two authenticated encryption algorithms and the hash algorithms. Table 16 lists a set of test vectors for different combinations of empty or nonempty AD and message. Table 17 lists a test vector for a pair of an AD and a message. Table 15 lists test vectors for a nonempty message and the empty message.

**Sycon permutation.** The input and output after 12 rounds of the permutation are:

**Input:**
00000000000000000000000000000000000000000000
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

**Output:**
347654D629FC982F0F085496C5612910670008CA
C78234FCC7743998FB35A737384442DDB1E9F498

Table 14: Step-by-step input-outputs of the permutation for 12 rounds

| |
|---|
| 00000000000000000000000000000000000000000000AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA |
| 575555BF020000EAAAFE5755FFAB020050FFFFFFFFFFFFFFFFFFFFFFFFFFFF5555555555555555 |
| DC075F5875055F17D7FBA2AA07A4E24723E5AA5483257A54EA076A0540ADC0AF9704EDDF3DAE4775 |
| 6DD1BD1281961443C862E256306CAE761D9411A81A5652B2F04DD274F25F62BD49806561AD842AE7 |
| 20CFDA31715EA6FF341CC603A4D2FA1775C61E155D50AB50A06D7EA9636D3B3ED38BB1C831F82111 |
| 0F7631EE64765DCA3BFD95583E7AA5B8FCB34E476FCA3392049CE1564C533D48C38B20868BEE6C96 |
| 031AF2926118A7AB0ACBCD9E71083F063E98836A0B00DF64DD48B0BD4BEE77518263988CB1B15188 |
| 5E2107360E5110648263DAA11CA3B8A0A030E334C55CA15A27D8A0A8CB443B1DA68D757990BCBBED |
| 480B2B1D19EFC0B6835B9576AA9A091B45EA0B7534222D10BB3407BCAA1EBE897A565959ABB548BC |
| CD746EF5623C07A81024EDAF8A057B113CAE114151358AC63442AC9BDAB15EFA8897C5B38AC1EA0D |
| 3A2C5CB10A72A0CD663E012E5C1DAD9F9286F5466CB3114011265EBEE6DE28442589CAF96649EB62 |
| F3C0ED761E8CEB13347C8C40E66168B7795485B4C59C3D971A990E082C5286A690D2A3BCC85E20A7 |
| 347654D629FC982F0F085496C5612910670008CAC78234FCC7743998FB35A737384442DDB1E9F498 |

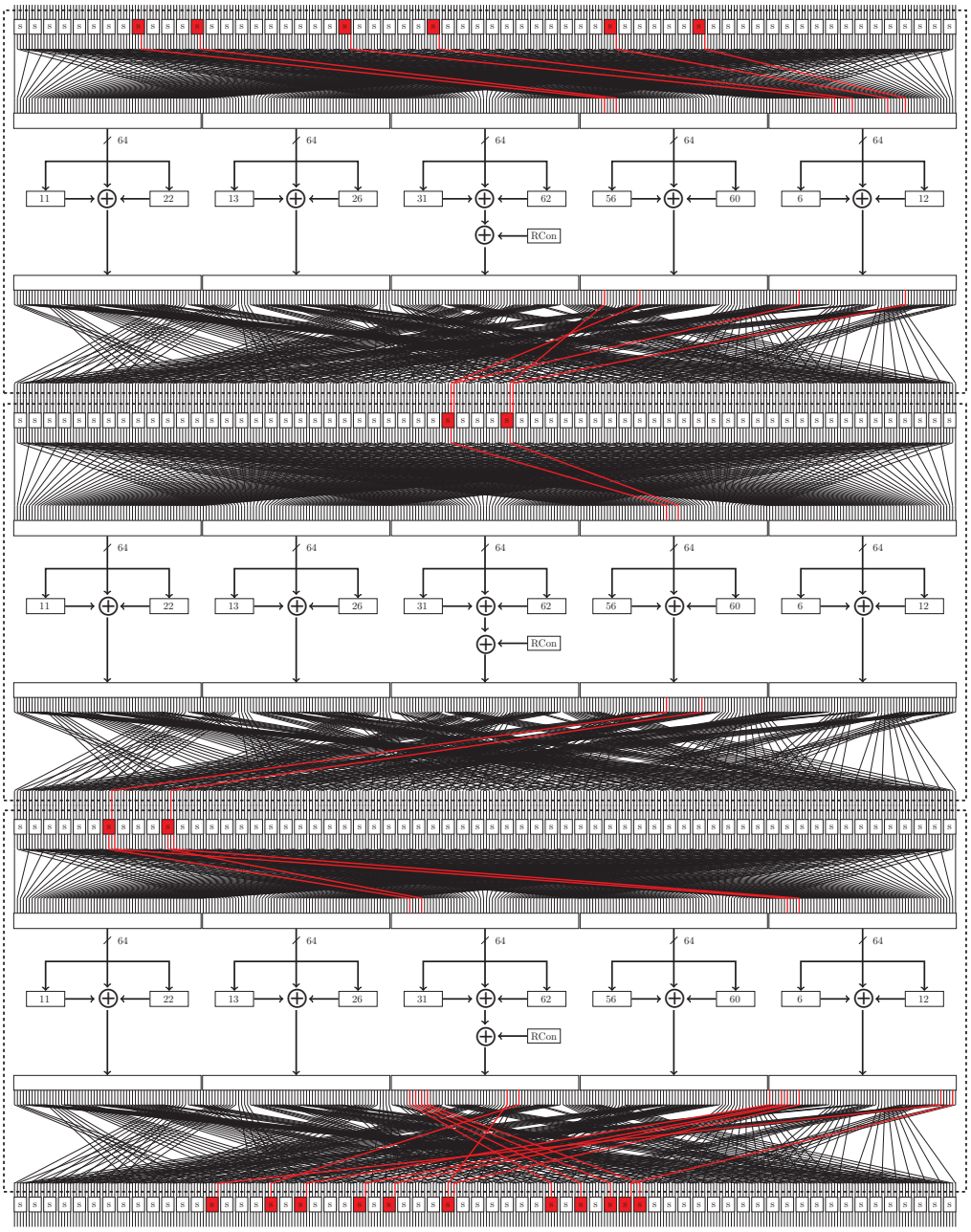



Figure 9: Sycon v1.0 has the 4-round differential trail whose number of active S-boxes is only 21.

Table 15: Test vectors for Sycon-Hash-64

| | |
|---|---|
| | **Message is nonempty** |
| Plaintext | "To authenticate, or not to authenticate" |
| Plaintext (byte) | 546F2061757468656E7469636174652C206F72206E6F7420746F2061757468656E746963617465 |
| Digest | 236B1F60C9871D0D5894E2FEFE12515D3B391E4A70FA70C1C7935325A103BCEC |
| | **Message is empty** |
| Plaintext | *empty* |
| Digest | 2B04CA328D41EC9EA39FCE4D9053932F86E2183C12F57DC72FAC7B28C1B8A2B0 |

Table 16: Test vectors for Sycon-AEAD-64

| | |
|---|---|
| | **Both message and AD are nonempty** |
| Key | 000102030405060708090A0B0C0D0E0F |
| Nonce | 000102030405060708090A0B0C0D0E0F |
| Associated data | 05AE023DC3105DA62894A16A0E260956 |
| Plaintext | "To authenticate, or not to authenticate" |
| Plaintext (byte) | 546F2061757468656E7469636174652C206F72206E6F7420746F2061757468656E746963617465 |
| Ciphertext | 12B0BA35C8A86C1E99CBAA08155E188E99C85AA82A1967D6745DBC0195F69F9FAC0942D7F233C4 |
| Tag | 62140577EFC0C8741F55BF5CB6E7851B |
| | **Message is nonempty** |
| Key | 000102030405060708090A0B0C0D0E0F |
| Nonce | 000102030405060708090A0B0C0D0E0F |
| Associated data | *empty* |
| Plaintext | "To authenticate, or not to authenticate" |
| Plaintext (byte) | 546F2061757468656E7469636174652C206F72206E6F7420746F2061757468656E746963617465 |
| Ciphertext | C38D58B95C04BD445BE10F1418E794EEF2A59DAFFC9D9F864D79FFE6594D68BF58FF2306E7CB25 |
| Tag | 80922236EB36EF3286EF768D31B4298F |
| | **AD is nonempty** |
| Key | 000102030405060708090A0B0C0D0E0F |
| Nonce | 000102030405060708090A0B0C0D0E0F |
| Associated data | 05AE023DC3105DA62894A16A0E260956 |
| Plaintext | *empty* |
| Ciphertext | *empty* |
| Tag | 154C33E2B768A74E18C0E1C90E266825 |
| | **Both AD and message are empty** |
| Key | 000102030405060708090A0B0C0D0E0F |
| Nonce | 000102030405060708090A0B0C0D0E0F |
| Associated data | *empty* |
| Plaintext | *empty* |
| Ciphertext | *empty* |
| Tag | 87D93CB5442AC330EBB37DA338D6CDC9 |

Table 17: Test vectors for Sycon-AEAD-96

| | |
|---|---|
| Key | 000102030405060708090A0B0C0D0E0F |
| Nonce | 000102030405060708090A0B0C0D0E0F |
| Associated data | 05AE023DC3105DA62894A16A0E260956 |
| Plaintext | "To authenticate, or not to authenticate" |
| Plaintext (byte) | 546F2061757468656E7469636174652C206F72206E6F7420746F2061757468656E746963617465 |
| Ciphertext | F2EB484C06258D6CEAC157A8360656AE50BBC1636028373A285D8B1E3FDFA7C803803ADDA4B1F1 |
| Tag | 30795DBE0B22B4AA2CF2880222A36946 |

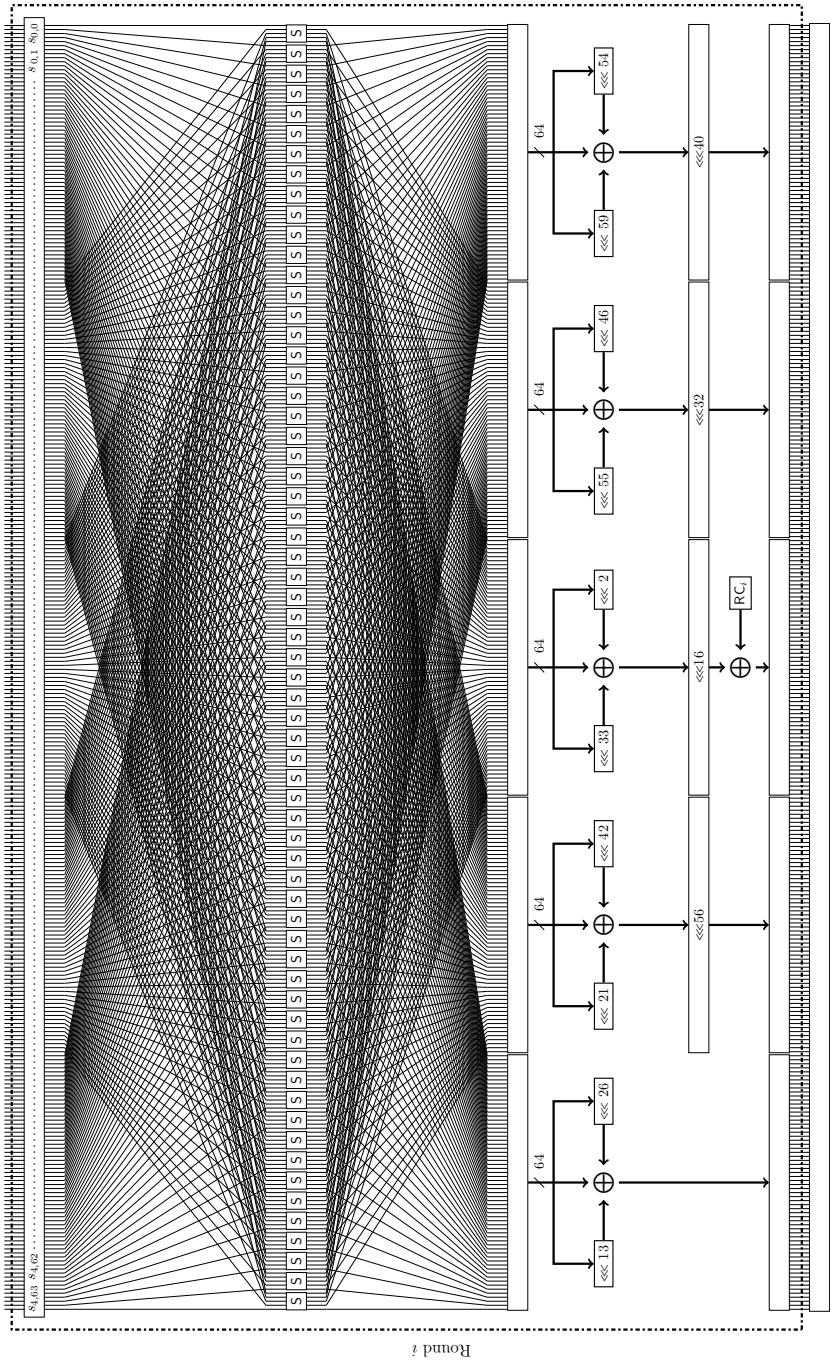



Figure 10: The holistic view of the round function of the Sycon permutation