Automatic Preimage Attack Framework on Ascon Using a Linearize-and-Guess Approach

Huina Li[2,1], Le He[1]✉, Shiyao Chen[3,1]✉, Jian Guo[1] and Weidong Qiu[2]✉

[1] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
[2] School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
[3] Strategic Centre for Research in Privacy-Preserving Technologies and Systems, Nanyang Technological University, Singapore, Singapore
{lihuina,qiuwd}@sjtu.edu.cn, {le.he,shiyao.chen,guojian}@ntu.edu.sg

Abstract. Ascon is the final winner of the lightweight cryptography standardization competition (2018 – 2023). In this paper, we focus on preimage attacks against round-reduced Ascon. The preimage attack framework, utilizing the linear structure with the allocating model, was initially proposed by Guo *et al.* at ASIACRYPT 2016 and subsequently improved by Li *et al.* at EUROCRYPT 2019, demonstrating high effectiveness in breaking the preimage resistance of Keccak. In this paper, we extend this preimage attack framework to Ascon from two aspects. Firstly, we propose a linearize-and-guess approach by analyzing the algebraic properties of the Ascon permutation. As a result, the complexity of finding a preimage for 2-round Ascon-Xof with a 64-bit hash value can be significantly reduced from 2^{39} guesses to $2^{27.56}$ guesses. To support the effectiveness of our approach, we find an actual preimage of all '0' hash in practical time. Secondly, we develop a SAT-based automatic preimage attack framework using the linearize-and-guess approach, which is efficient to search for the optimal structures exhaustively. Consequently, we present the best theoretical preimage attacks on 3-round and 4-round Ascon-Xof so far.

Keywords: Ascon · Preimage Attack · SAT

1 Introduction

Ascon [DEMS21] was selected as the final winner of the lightweight cryptography standardization competition in February 2023, including AEAD and Hash schemes. The hash function uses the sponge construction to map arbitrarily long inputs into fixed length (Ascon-Hash), or extendable (Ascon-Xof) outputs. The underlying permutation of Ascon hash functions consists of 12 iterations of a quadratic round function executed on a 320-bit state. In this paper, we mainly focus on the preimage attacks on round-reduced Ascon-Xof.

Currently, there are very few results that find preimages on the round-reduced Ascon. Dobraunig *et al.* [DEMS19] first found a preimage on Ascon-Xof (the length of the hash value is truncated to 64 bits) up to 2 rounds with 2^{39} guesses. In [QHD+23], Qin *et al.* presented a generic framework of Meet-in-the-Middle (MitM) preimage attack on sponge-based hashing with the help of bit-level Mixed Integer Linear Programming (MILP) model, which extends up to 4 rounds of Ascon. However, their model does not guarantee optimal solutions with respect to attack complexities. More recently, Qin *et al.* [QZH+23] extended the MitM attack framework into collision attacks and also developed a set of techniques, such as *weak-diffusion structure* and a heuristic searching strategy, to improve previous preimage attacks [QHD+23], but required a huge amount of memory.

In this paper, we aim at improving preimage attacks on the round-reduced Ascon-Xof. Our attacks inherit many ideas from existing researches, but apply them in new ways.

Linear Structure. Generally, in preimage attacks based on the linearization technique, the objective is to construct algebraic systems with high degrees of freedom. Within this approach, a larger number of remaining degrees of freedom indicates a more effective preimage attack, as it significantly decreases the complexity associated with solving the entire algebraic system. Especially, at ASIACRYPT 2016, an important linearization technique – *linear structure*, for improving preimage attacks on round-reduced Keccak, was introduced by Guo *et al.* [GLS16]. Building upon the 1.5-round linear structures of Keccak-224/256, they presented the first practical preimage attacks on 2-round Keccak-224/256. Later, at EUROCRYPT 2019, Li and Sun [LS19] proposed a 2-block model named allocating model to further improve the linear structure. With the allocating model, more degrees of freedom can be left in their structure after 2-round entire linearization. Subsequently, several preimage attacks on round-reduced Keccak based on linear structures with the allocating model were presented [Raj19, LIMY20, LHY21, HLY21, HLY22, LHY23].

Inspired by their thoughts, we propose a linearize-and-guess approach by exploiting structural properties of Ascon's internal permutation, namely, the limited diffusion of its internal mappings and statistical properties of its substitution layer, such as the probabilistic equation $b_0 = a_3 + a_2 + a_0$ [1] of substitution layer that holds with a probability of $\frac{3}{4}$. Consequently, we find an actual preimage in 2 rounds of Ascon-Xof with a 64-bit hash value. In comparison to the linear structure technique [GLS16], our approach is also based on linearization thought, but allows the presence of non-linear bits [2] (quadratic, even quartic bits) within the structure rather than linearizing the entire structure. Our approach is similar to the previous methods [Raj19, LIMY20, HLY22, LHY23] used on the Keccak's preimage attack that allows non-linear parts to exist in the structure and just generate output linear equations on linear parts. However, their methods are only applicable to the cases that the algebraic degrees of non-linear parts are at most 2.

Automated Tools. Previous attacks based on linear structures were manually constructed [GLS16, LSLW17, LS19, Raj19, LIMY20, LHY21, HLY21, HLY22, LHY23], typically imposing constraints on the input variables to ensure they belong to the so-called column-parity-like (CP-like) kernel. However, for Ascon, its linear layer is highly flexible (independently rotate each 64-bit word by different offsets) and its non-linear layer is more complex, which gives larger challenges in manually designing a longer structure. The most popular automated tools are the Boolean satisfiability problem (SAT) [MP13] and Mixed Integer Linear Programming (MILP) [MWGP11]. The main difference between them is the forms of describing constraints and objective functions, such as clause normal forms (CNF) in SAT tools and linear inequality forms in MILP tools.

In recent years, a series of automatic tools based methods have been introduced to boost the preimage attacks. In [MS13], Morawiecki *et al.* proposed a practical preimage attack based on SAT tools on a simplified Keccak using different parameters rather than the recommended ones. In [BDG$^+$21, BGST22], Bao *et al.* introduced the MILP-based automatic search framework for MitM preimage attacks on AES-like hashing. At EUROCRYPY 2023, Qin *et al.* [QHD$^+$23] applied the MitM preimage attacks on sponge-based hashing and introduced a generic bit-level MILP-based MitM preimage attack framework on Keccak, Ascon, and Xoodyak. However, it is not clear whether Qin *et al.*'s model is efficient for exhaustively searching the optimal attacks on Ascon. Different from Qin *et al.*'s MILP-based preimage attack frameworks, we establish a SAT-based automatic preimage attacks framework using the linearize-and-guess approach on Ascon-

[1] The substitution layer of Ascon permutation can be regarded as 64 S-boxes with 5 inputs $(a_0, a_1, a_2, a_3, a_4)$ and 5 outputs $(b_0, b_1, b_2, b_3, b_4)$.

[2] Non-linear bit, corresponds to a bit whose polynomial expression contains product terms of unknown variables (the algebraic degree is at least 2).

Xof, as SAT is well-suited for bitwise descriptions of behavior. Leveraging SAT models and the off-the-shelf `CaDiCaL` solver, it becomes efficient to exhaustively search for optimal preimage attacks. Both Qin *et al.*'s attack and our attack are structure-based preimage attacks, but exhibit notable distinctions. Qin *et al.*'s attack strategy involves dividing the variables of the initial state into two distinct classes. While variables within each class can be multiplied together, the multiplication of variables from different classes is not permitted. From this aspect, their MILP model becomes quite complex as they have to describe the relationships between different classes, which potentially affects the efficiency of finding solutions. In contrast, our attack employs a single class where all variables can be multiplied together. However, to establish more linear equations leaked by the hash value, we need to cost extra degrees of freedom to linearize non-linear bits within the structures, particularly in the case of 4 rounds. Furthermore, our tool does not require any memory usage.

Our Contributions. Through comprehensive analysis of the structural properties of Ascon, we find that by applying special linear or quadratic conditions to different substitution layers, the number of degrees of freedom left can be effectively controlled and increased. Additionally, we observe a crucial statistical property of the substitution layer, namely, the probabilistic equation $b_0 = a_3 + a_2 + a_0$ holding with a probability of $\frac{3}{4}$. Leveraging this observation, we can establish linear equations based on some fixed hash bits, as long as the algebraic degrees of $a_3, a_2,$ and a_0 are at most 1. This releases the restrictions in constructing linear equations leaked by the hash value. In this paper, we formalize these ideas and present a linearize-and-guess approach for searching $(n-1)$-round structures used in n-round preimage attacks. This approach leads to preimage attacks on 2-round Ascon-Xof with reduced complexities from 2^{39} guesses [DEMS19] to $2^{27.56}$ guesses.

Besides, we extend our approach to develop an automatic preimage attack framework for searching the optimal preimage attacks with SAT model. Our framework considers various model parameters, such as the remaining degrees of freedom, the number of linear equations leaked by the hash value, and the search complexity of finding a restricted inner part, as well as the size of random space. To further speed up the search, we also incorporate the rotational symmetry property of Ascon permutation into our model. Consequently, improved preimage attacks on 3-round and 4-round Ascon-Xof with 128-bit hash outputs are achieved. The previous attacks and our new results are summarized in Table 1. The source codes are available at `https://github.com/HuinaLi/Automatic-Preimage-Attack-Framework-on-Ascon-Xof`.

Organization. This paper is organized as follows. In Section 2, we give some preliminaries and notations of Ascon, and briefly describe the related techniques used in previous preimage attacks. In Section 3, we present our linearize-and-guess approach. In Section 4, we introduce our SAT-based preimage attack framework. Its application to 3-round and 4-round Ascon-Xof are provided in Section 5 and in Section 6, respectively. Our conclusions are drawn in Section 7.

2 Preliminaries

2.1 Description of Ascon

Ascon [DEMS21] was announced by NIST as the final winner in the lightweight cryptography standardization competition on February 7th, 2023. The Ascon family consists of AEAD and Hash schemes. Both schemes operate on a 320-bit state which updates with permutations p^a (12 rounds) and p^b (6 rounds). Each bit is represented as $A[y][x]$, where $0 \leq y < 5$, $0 \leq x < 64$. The state A is arranged into 5 rows or 64 columns. Each row is a

Table 1: Summary of preimage attacks on Ascon-Xof. Hash: the length of the digest in bits. Size: the number of linear equations leaked by the hash value. Guesses: the number of required solutions. Solving Time: the average complexity of obtaining a single solution.

Round	Hash	Size	Guesses	Solving Time†	Final Time	Memory	Reference
2	64	25	2^{39}	$2^{-0.04}$	2^{39}	-	[DEMS19]
		64	$2^{27.56}$	2^{4}	$2^{31.56}$	-	Section 3
3	128	-	-	-	$2^{120.58}$	2^{39}	[QHD+23]
		-	-	-	$2^{114.53}$	2^{30}	[QZH+23]
		27	$2^{112.205}$	$2^{-0.29}$	$2^{112.205}$	-	Section 5
4	128	-	-	-	$2^{126.4}$	2^{45}	[QHD+23]
		-	-	-	$2^{124.67}$	2^{50}	[QZH+23]
		6	$2^{124.49}$	$2^{-1.01}$	$2^{124.49}$	-	Section 6

†Here 'Solving Time' is determined by the ratio of the number of bit operations required for one Gaussian Elimination turn to the number of bit operations in round-reduced Ascon.

64-bit register word, denoted by $A[y][\star]$ (as shown in Figure 1). Each column can be seen as a 5-bit S-box, for example, $(a_0, a_1, a_2, a_3, a_4)$ where a_0 indicates the most significant bit (MSB).

The round function consists of three steps p_C, p_S, and p_L, denoted by $p = p_L \circ p_S \circ p_C$.

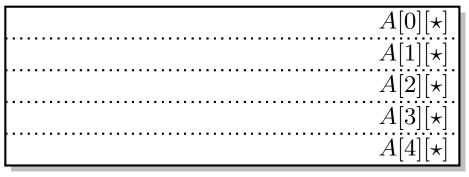


Figure 1: Illustration of the Ascon state.

Addition of Constants (p_C). p_C adds a round constant c_i to register word $A[2][\star]$ in round i. We ignore the details of c_i here since it does not affect our attacks to be presented.

Substitution Layer (p_S). p_S operates on the state A with 64 parallel applications of the 5-bit S-box to each bit-slice of the five register words, which is affine equivalent to the non-linear operation, *i.e.*, χ mapping of Keccak. We suppose the inputs of one S-box are $A[\star][x] = (a_0, a_1, a_2, a_3, a_4)$, and the outputs are $(b_0, b_1, b_2, b_3, b_4)$. The algebraic normal form (ANF) of the S-box is as follows:

$$\begin{aligned}
b_0 &= a_4a_1 + a_3 + a_2a_1 + a_2 + a_1a_0 + a_1 + a_0\\
b_1 &= a_4 + a_3a_2 + a_3a_1 + a_3 + a_2a_1 + a_2 + a_1 + a_0\\
b_2 &= a_4a_3 + a_4 + a_2 + a_1 + 1\\
b_3 &= a_4a_0 + a_4 + a_3a_0 + a_3 + a_2 + a_1 + a_0\\
b_4 &= a_4a_1 + a_4 + a_3 + a_1a_0 + a_1
\end{aligned} \tag{1}$$

Linear Diffusion Layer (p_L). p_L provides diffusion within each 64-bit register word $A[y][\star]$ as follows, where the first offset is denoted by $r_0 \in \{19, 61, 1, 10, 7\}$, and the second one is denoted by $r_1 \in \{28, 39, 6, 17, 41\}$.

$$\begin{aligned}
A[0][\star] &= B[0][\star] + (B[0][\star] \ggg 19) + (B[0][\star] \ggg 28)\\
A[1][\star] &= B[1][\star] + (B[1][\star] \ggg 61) + (B[1][\star] \ggg 39)\\
A[2][\star] &= B[2][\star] + (B[2][\star] \ggg 1) + (B[2][\star] \ggg 6)\\
A[3][\star] &= B[3][\star] + (B[3][\star] \ggg 10) + (B[3][\star] \ggg 17)\\
A[4][\star] &= B[4][\star] + (B[4][\star] \ggg 7) + (B[4][\star] \ggg 41)
\end{aligned} \tag{2}$$

Ascon Hash Scheme. The Ascon hash scheme including Ascon-Hash and Ascon-Xof, is built upon the p^a permutation using the sponge construction to map arbitrarily long inputs into fixed length outputs and extendable outputs, respectively.

The sponge construction illustrated in Figure 2 works on a 320-bit state, which is split into two parts: the first part called the outer part contains the r bits rate of the state ($r = 64$) and the second part named the inner part contains the $c = 320 - r$ bits capacity of the state. Firstly, Ascon-Hash and Ascon-Xof initialize a 320-bit initial state using the 12-round permutation p^a. Then, the sponge construction processes the message M in two phases. In the absorbing phase, the message M firstly appends a single 1 and a smallest number of 0s such that the length of the padded message is a multiple of r. After padding, M is split into n blocks of r-bit *i.e.*, $M_1||\ldots||M_n$. These message blocks are processed iteratively by XORing each block into the first r bits of the current state, and then applying the p^a permutation on the value of the 320-bit state. After processing all the blocks, the sponge construction switches to the squeezing phase. In the squeezing phase, the d output bits are produced iteratively. In each iteration the first r bits are extracted from the state and the permutation p^a is applied.

In this paper, we focus on Ascon-Xof with a 128-bit hash value and a 128-bit security claim against preimage attack.

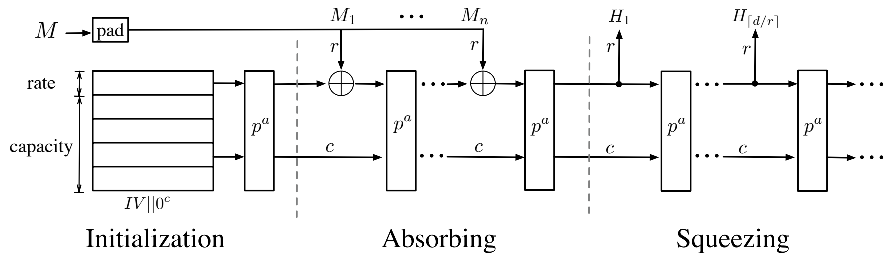


Figure 2: The illustration of the Ascon hash scheme.

2.2 Using Linear Structure to Construct Linear System

The linear structure technique was first formalized and developed by Guo *et al.* [GLS16] in 2016. Based on this novel technique, a series of cryptanalysis records of preimage attacks against Keccak and zero-sum distinguishers of the underlying permutation Keccak-f were broken. The main idea is to linearize the whole state after several rounds with sufficient degrees of freedom left. Figure 3 gives a 1.5-round linear structure of Keccak-f[1600] with 256 degrees of freedom left.

Let A^0 be a starting state, and each bit is defined as $A^0[x][y][z]$ ($0 \le x < 5, 0 \le y < 5, 0 \le z < 64$). $A^0[x][y][z]$ is either a linear bit that includes the linear polynomial of variables (the algebraic degree is 1) or a constant bit. Suppose six lanes indexed by $A^0[x][y][\star]$ where $x = 0, 2$ and $y = 0, 1, 2$ are linear bits (*i.e.*, each specified lane has 64 linear bits). To limit these linear bits not be diffused by θ mapping, 128 linear conditions are set up such as $A^0[x][2][z] = \sum_{y=0}^{1} A^0[x][y][z] + c_{x,z}$ holds where $c_{x,z} = 0$ or $c_{x,z} = 1$, such that these linear bits in each column sum to a constant. Note that each linear condition can be ensured by spending 1 degree of freedom. Thus, the remaining degrees of freedom are given by $384 - 128 = 256$.

In Figure 3, we can observe how the propagations of the linear bits pass through the round function $R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$ of KECCAK-f[1600]. Each bit of the lanes with yellow represents a linear bit. The other lanes are all constants where gray and white bits stand for values 0 and arbitrary constants, respectively. Note that ρ and π linear mappings only re-arrange the location of the linear bits, and ι mappings add one round constant, so the linear bits remain undiffused by these linear mappings. The algebraic degree of the non-linear mapping χ is 2, and only when all linear bits before χ are not adjacent to each other, the algebraic degree of each output bit keeps at most 1. Thus, it makes sure that the algebraic degree of the entire state remains at most 1 after 1.5 rounds.

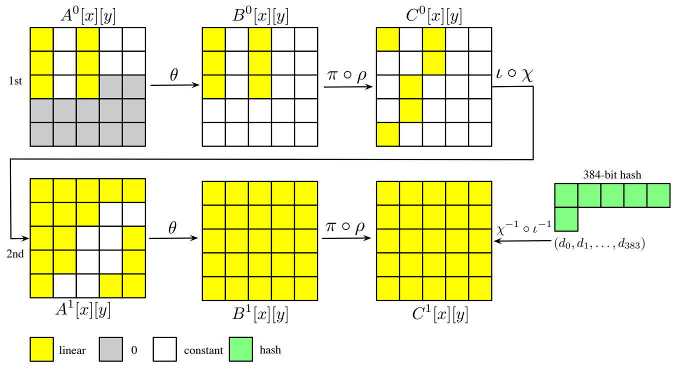


Figure 3: 1.5-round linear structure with 256 degrees of freedom left used in preimage attack on 2-round KECCAK-384.

According to [GLS16], each row with 5-bit output (hash) is corresponding to 5 linear equations. In this case, we could construct a linear system with 256 equations and 256 degrees of freedom (each system has on average one solution). Counting the padding rule, thus the final search complexity of preimage attack on 2-round KECCAK-384 is $2^{384-256+1} = 2^{129}$. For a complete specification of KECCAK and linear structures, we refer the interested reader to the original specification [BDPA13, GLS16].

2.3 Linear Structure with the Allocating Model

Since in sponge constructions of KECCAK a large portion of the initial state is fixed and cannot be chosen by the cryptanalyst, such restrictions (the inner part of the initial state must be all '0') become the bottleneck of 1-block preimage attacks with the linear structure. To break this bottleneck, at EUROCRYPT 2019, Li et al. [LS19] introduced the allocation approach to multi-block preimage attacks on round-reduced KECCAK-224/256, which releases the constraints on the initial state of the linear structure and leaves more degrees of freedom. In their model, a 2-round linear structure with 194 degrees of freedom left is utilized, resulting in a significant improvement in the theoretical complexity of preimage attacks on 3-round and 4-round KECCAK-224/256. The whole complexity of preimage attacks is divided into two stages:

1. Precomputation Stage: Find an inner part that satisfies certain conditions for the initial state of the linear structure. Let 2^{d_1} be the search complexity of finding this restricted inner part.

2. Online Stage: Construct the algebraic systems according to the linear structure that includes linear equations and nonlinear equations leaked by the given hash value,

and then solve these systems for finding a multi-block preimage.

Random Space of a Single Structure[3]**.** The maximum number of different algebraic systems that a single structure can generate is determined by the random space of this structure. We denote the size of random space as 2^{d_r} in our preimage attacks on Ascon. Since any 1-bit difference in the initial state will significantly affect the final hash value, this number can be well borrowed to evaluate the size of random space, *i.e.*, the number of all possible cases of the initial state. Suppose the capacity part is fixed and d_f out of r bits are set as variables. Then, merely $r - d_f - 1$ constant bits (excluding one padding bit for Ascon) can assign as 0 or 1, thus the size of random space is calculated as $2^{d_r} = 2^{r-d_f-1}$.

The Final Gain. Given an algebraic systems according to the structure that includes d_e linear equations and $d - d_e$ non-linear equations leaked by the given hash value with d_f degrees of freedom left ($d_f \geq d_e$), where d is the length of the hash value and d_e is the number of linear equations leaked by the hash value. Then, the final gain is calculated as 2^{d_e}. That means the final search complexity of finding a preimage is 2^{d-d_e}.

The Final Complexity. The final time complexity is determined by the complexity of the following two parts.

1. The final search complexity of finding a preimage[4], denoted by 2^{d_2}.
   If we ignore the padding bits, $2^{d_2} = 2^{d-d_e}$. Since one known structure can only generate 2^{d_r} linear systems, we need to re-start $2^{d_2-d_r}$ structures. In other words, $2^{d_2-d_r}$ restricted inner parts from previous message blocks need to be generated. And the search complexity of finding such an inner part is 2^{d_1}. Thus, the final search complexity is calculated as:

   $$\max\{2^{d_1} \times 2^{d_2-d_r},\ 2^{d_2}\}$$

   A special case should be noted: in the allocating model, it is required that $2^{d_r} \geq 2^{d_1}$. When implementing the allocation approach, it is crucial to ensure that the size of the random space satisfies $2^{d_r} \geq 2^{d_1}$, to guarantee a final search complexity of 2^{d_2}. Otherwise, the final search complexity becomes $2^{d_1} \times 2^{d_2-d_r}$, greater than 2^{d_2}.

2. The average complexity of obtaining a single solution, denoted by T_s.
   T_s is computed by the ratio of the number of bit operations required for one Gaussian Elimination turn to the number of bit operations in round-reduced Ascon. Besides, there is still a verification time, denoted by T_v. For each solution to examine whether it can match the given hash value, where T_v is always 2^0 (once target primitive call).

Thus, the final complexity of preimage attack is summarized in Equation 3. When T_s is less than 2^0, the final complexity is 2^{d_2}.

$$2^{d_2} \times (T_s + T_v) \tag{3}$$

2.4 SAT-based Cryptanalysis

Given a Boolean formula $f(x_1, x_2, \ldots, x_n)$, the Boolean satisfiability (SAT) problem is to determine whether there is any assignment of values to these Boolean variables which makes the formula true. The Boolean formula is satisfiable if a valid assignment exists, otherwise it is unsatisfiable. A formula in CNF consists of one or more clauses joined by *conjunctions* ($\wedge$), where each clause is a *disjunction* ($\vee$) of *literals*, each literal represents a

[3]In [HLY22], they gave a more formal definition of the random space of a single structure with the allocating model on Keccak.

[4]The final search complexity is also called the total of guesses, which is equal to the number of required solutions of the preimage attack.

positive or negative variable, *e.g.*, x_i or $\neg x_i$. Most of the previously introduced SAT-based cryptanalysis methods [SWW18, SWW21, GLST22] encode directly the cryptanalysis problem in CNF as a SAT instance under the Markov assumption and then invoke the off-the-shelf SAT solver to solve it. There are many off-the-shelf SAT solvers available which have been introduced into cryptanalysis, such as `CryptoMiniSat` [SNC09] and `CaDiCaL` [Arm19]. These solvers based on *conflict-driven clause learning* (CDCL) [MLM21] support CNF (in DIMAC format) as their input. In this paper, we choose `CaDiCaL` as the SAT solver due to its simple installation process and user-friendly interface.

3 The Linearize-and-Guess Approach

In this section, we present a linearize-and-guess approach. This approach allows partial linearization of the underlying permutation of Ascon for up to three rounds with high degrees of freedom. Our main idea is to search for $(n-1)$ round optimal structures used in the preimage attack on n-round Ascon-Xof, where $n \leq 4$. With these structures, more linear equations are set up and lower final search complexity can be obtained. More impressively, we observe that specific conditions within different substitution layers can efficiently control the diffusion of linear bits. Besides, we observe an important statistical properties of substitution layer p_S, which significantly releases the restrictions in setting up linear equations leaked by the hash value. These key observations make it possible to find longer optimal structures, such as 3-round structures.

Next, more details of the linearize-and-guess approach are presented. To show the effectiveness of our approach, we give an improved preimage attack on 2-round Ascon-Xof in this section.

3.1 Diffusion of Linear Bits through the Substitution Layer p_S

We start by studying the algebraic properties of the substitution layer p_S of each forward permutation, which consists of 64 parallel S-boxes. Consider a single S-box with 5 input bits denoted by $(a_0, a_1, a_2, a_3, a_4)$, and corresponding 5 output bits denoted by $(b_0, b_1, b_2, b_3, b_4)$. Both the input and output bits are represented as polynomials of unknown message variables and constants according to Equation 1. For each input or output bit, we classify it into one of three categories based on its polynomial expression. This classification method allows us to analyze the diffusion of linear bits during the p_S operation more conveniently, providing clearer insights into the process.

- Constant bit, refers to a bit that does not contain any unknown variables, denoted by ‘c’.
- Linear bit, corresponds to a bit whose polynomial expression only contains XOR combinations of constants and unknown variables, denoted by ‘v’.
- Unknown bit, corresponds to a bit whose polynomial expression contains product terms of unknown variables, denoted by ‘q’.

Our objective is to find some specific conditions imposed on the inputs of i-th p_S $(1 \leq i < 4)$ that can significantly control the diffusion of linear bits, *i.e.*, leading to the outputs containing as many as possible constant bits and linear bits.

p_S in the 1-st Round. According to sponge construction, one message block is processed by XORing the first $r = 64$ bits of the current state, such that in the first p_S, only a_0 may be a linear bit.

Thus, 5 diffusion patterns in the first p_S are concluded in Table 2. Those patterns are distinguished by different linear conditions. For instance, when the last four bits of the inputs are all constant bits, if two linear conditions are satisfied, such as $a_1 = 0$ and $a_3 + a_4 = 1$, then the first two bits of 5-bit outputs are linear bits.

Table 2: Diffusion patterns in the first p_S. # C: represents the number of required conditions for each diffusion pattern.

Inputs	Outputs	Condition	$\#C$	Inputs	Outputs	Condition	$\#C$
ccccc	ccccc	-	0	vcccc	vvccc	$a_1 = 0; a_3 + a_4 = 1$	2
vcccc	vvcvc	$a_1 = 0; a_3 + a_4 = 0$	2	vcccc	cvccv	$a_1 = 1; a_3 + a_4 = 1$	2
vcccc	cvcvv	$a_1 = 1; a_3 + a_4 = 0$	2				

Property 1. *Given a S-box with 5-bit inputs (a_0,a_1,a_2,a_3,a_4) and 5-bit outputs (b_0, b_1,b_2,b_3,b_4), if a_0 is a linear bit and other four bits are all constant bits, then b_2 must be a constant bit.*

Proof. According to the ANF of the S-box: $b_2 = a_4a_3 + a_4 + a_2 + a_1 + 1$, b_2 is irrelevant to a_0. Hence, b_2 must be a constant bit.

p_S in the 2-nd Round. Due to Property 1 and the independent calculation of p_L within each row, the third input bit a_2 of the second p_S must be a constant bit. We summarize all possible 16 diffusion patterns that are used in the second p_S of 2-round structures in Table 3. However, if we employ these 16 diffusion patterns in the second p_S of 3-round structures, this would result in linear bits hardly existing within the outputs of the third p_S. Therefore, to overcome this problem, some specific conditions have to be introduced to obtain the expected outputs with more constant bits, since these constant bits can inhibit the generation of unknown bits to some extent. We observe that there are 6 diffusion patterns in the second p_S of 3-round structures that can effectively prevent the generation of unknown bits, as shown in Table 3. The last two candidate diffusion patterns ($vvccc, qvvvq$) and ($cvccv, qvvvq$) are based on the settings that we only allow unknown bits ('q') to exist in the first bit b_0 or the last bit b_4 in the outputs of the second p_S (also holds in the inputs of the third p_S). Next, we will elucidate the reasons behind our choice of this particular settings.

Table 3: Diffusion patterns in the second p_S. # C: represents the number of required conditions for each diffusion pattern. Used in: the diffusion patterns are used in the second p_S operation of n-round structure, $n = 2, 3$.

Used in	Inputs	Outputs	Condition	$\#C$	Inputs	Outputs	Condition	$\#C$
2-round	ccccc	vvvvv	-	0	vcccc	vvvvv	-	0
	cvccc	vvvvv	-	0	cccvc	vvvvv	-	0
	ccccv	vvvvv	-	0	vvccc	qvvvq	-	0
	cvccv	qvvvq	-	0	vccvc	vvvqv	-	0
	cvcvc	vqvvv	-	0	vvcvc	qqvqq	-	0
	vcccv	vvvqv	-	0	vvccv	qvvqq	-	0
	cccvv	vvqvv	-	0	vccvv	vvqqv	-	0
	cvcvv	qqqvq	-	0	vvcvv	qqqqq	-	0
3-round	ccccc	ccccc	-	0	vcccc	vvccc	$a_1 = 0; a_3 + a_4 = 1$	2
	vcccc	cvccv	$a_1 = 0; a_3 + a_4 = 1$	2	cvccc	ccvvc	$a_2 = 0; a_3 = 1; a_0 + a_4 = 1$	3
	ccccv	cvccv	$a_0 = 1; a_1 = 0; a_3 = 1$	3	ccccv	vvccc	$a_0 = 1; a_1 = 1; a_3 = 1$	3
	vvccc	qvvvq	-	0	cvccv	qvvvq	-	0

Property 2. *Given a S-box with 5-bit inputs (a_0,a_1,a_2,a_3,a_4) and 5-bit outputs (b_0, b_1,b_2,b_3,b_4), if any single unknown bit is among a_1, a_2, or a_3, then one of b_3, b_2, b_0 must be an unknown bit.*

Proof. Let us analyze again the ANF of S-box. We can re-write b_2 as follows:

$$b_2 = a_4a_3 + a_4 + a_2 + a_1 + 1$$

If a_1 or a_2 is an unknown bit and other four bits are all linear or constant bits, it is obvious that b_2 must be an unknown bit. Similarly, since $b_0 = a_4a_1 + a_3 + a_2a_1 + a_2 + a_1a_0 + a_1 + a_0$, if a_3 is an unknown bit and other four bits are all linear or constant bits, it is obvious that b_0 must be an unknown bit.

p_S in the 3-rd Round. Before introducing this part, we recommend that the readers first understand how to generate guess linear equations leaked by the hash value in Subsection 3.2. In the third p_S operation, our objective is to maximize the presence of linear bits in b_3, b_2, and b_0 (these bits are the outputs of the third p_S operation), so that more guess linear equations leaked by the hash value from the outputs of the fourth p_S can be set up. Specially, we find that the position of a single unknown bit in the inputs determines whether b_3, b_2, b_0 can be linear bits according to Property 2.

Based on above analysis and observations, we only allow unknown bits ('q') exist in the first bit a_0 or the last bit a_4 in the inputs of the third p_S (in other words, a_1, a_2 and a_3 must be constant or linear bits). The diffusion patterns in the third p_S are summarized in Table 14 (Due to page limits, this table is provided in Section A.1).

3.2 Generate Guess Linear Equations from the Output of p_S

According to the sponge construction, the 128-bit hash of Ascon-Xof is generated iteratively, and each iteration involves the extraction of the first $r = 64$ bits of the state. In addition, since the linear diffusion layer p_L operates on five 64-bit words in parallel, the input of the last p_L, namely, the output of the last p_S can be recovered from each extraction. Therefore, for each S-box $(a_0, a_1, a_2, a_3, a_4)$, only the output bit b_0 is known, where a_i $(0 \leq i < 5)$ may be constant bits, linear bits, or unknown bits. In this subsection, we introduce three linearization ways to establish linear equations leaked by the hash value.

Observation 1. *If the algebraic degrees of all 5 input bits are at most 1, namely either linear bits or constant bits, and a_1 is a constant bit, then $(a_4+a_2+a_0+1)a_1+a_3+a_2+a_0 = b_0$ must be a linear equation.*

Observation 2. *If the algebraic degrees of all 5 input bits are at most 1, and a_4, a_2 as well as a_0 are all constant bits, then $(a_4 + a_2 + a_0 + 1)a_1 + a_3 + a_2 + a_0 = b_0$ must be a linear equation.*

Observation 3. *If the algebraic degrees of a_3, a_2, a_0 are at most 1, then the guess linear equation $a_3 + a_2 + a_0 = b_0$ holds with a probability of $\frac{3}{4}$ when 5 input bits are uniformly distributed.*

For a better understanding, we slightly give an explanation for Observation 3. Let us consider equation $b_0 = (a_4 + a_2 + a_0 + 1)a_1 + a_3 + a_2 + a_0$. Suppose all five input bits are linear bits. There are two ways to linearize this equation:

- By assigning either $a_1 = 0$ or $a_4 + a_2 + a_0 + 1 = 0$, the equation can be linearized. Together with equations from the assigned, we obtain two linear equations in total, and each equation can bring a gain of $2^{0.5}$.
- Note that the quadratic term $(a_4 + a_2 + a_0 + 1)a_1 = 0$ holds in fact with a probability of $\frac{3}{4}$. Thus, one linear equation $a_3 + a_2 + a_0 = b_0$ can be obtained by excluding the quadratic term, which can still bring a gain of $\frac{3}{4}/\frac{1}{2} \approx 2^{0.585}$.

In summary, one guess linear equation with a probability of $\frac{3}{4}$ can bring a larger gain.

3.3 1-round Linear Structure

In this section, we introduce a 1-round linear structure via the linearize-and-guess approach in Figure 4. The inner part of the initial state A^0 consists of 256 fixed constant bits generated from the previous absorbing phase. For the current message block $M_n, n = 2$, it only undergoes XOR operations with the first $r = 64$ bits of the initial state A^0, resulting in the variables confined solely to the first row $A^0[0][\star]$.

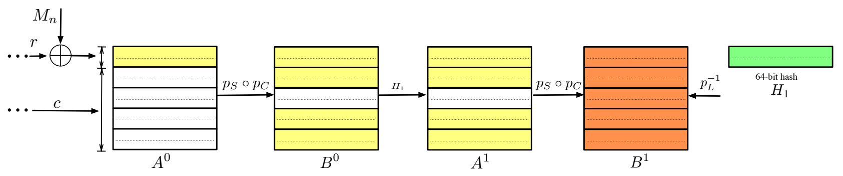


Figure 4: 1-round linear structure using the linearize-and-guess approach. Green stands for hash bits, yellow stands for linear bits, white stands for constant bits, and orange stands for either linear bits or unknown bits.

Improved 2-round Preimage Attack on Ascon-Xof. Our preimage attack is built upon a 1-round structure as shown in Figure 4, which can leave 64 degrees of freedom. By exploiting Observation 3, we construct a linear system that includes 64 guess linear equations with 64 degrees of freedom. The total gain is calculated as $2^{64\times 0.585} = 2^{37.44}$. In summary, we improve the search complexity of preimage attack on 2-round ASCON-XOF with a 64-bit hash output from 2^{39} to $2^{64-37.44+1} = 2^{27.56}$ guesses after dealing with paddings.

Practical Preimage Attack Experiment. The preimage attack on the 2-round ASCON-XOF is parameterized by $d_1 = 0$, $d_r = 0$, and $d_2 = 2^{27.56}$. With these parameters, we can proceed to mount a practical preimage attack. The attack follows a step-by-step procedure, iterating through $2^{d_2-d_r+d_1}$ different values of M_1. Firstly, an inner part of the initial state is randomly generated (from the outputs of M_1). Then, we construct an algebraic system with 64 guess linear equations (derived from 64 quadratic equations) leaked by the hash value, and solve them through Gaussian Emilination. If no solution is obtained, or the solution fails to pass the verification, the process continues by regenerating a new inner part. Finally, we present an actual preimage with a 64-bit hash of all '0' hash value in Table 4 to support our preimage attack on 2-round ASCON-XOF.

Table 4: An actual preimage of 2-round ASCON-XOF with a 64-bit hash value.

Initial inputs	Initial outputs	Absorb M_1
00400c0000000000	b57e273b814cd416	481bdfc24b87d0bb
0000000000000000	2b51042562ae2420	2b51042562ae2420
0000000000000000	66a3a7768ddf2218	66a3a7768ddf2218
0000000000000000	5aad0a7a8153650c	5aad0a7a8153650c
0000000000000000	4f3e0e32539493b6	4f3e0e32539493b6
M_1 outputs	Absorb M_2 (A^0)	M_2 outputs
930ed7e04024b01e	5d97d527a376a0bd	0000000000000000
71ed707bb8746da5	71ed707bb8746da5	0ad74802b2878fda
ee56d8257ccc2dd9	ee56d8257ccc2dd9	0ae07279817d7633
17faddf5e3f1251d	17faddf5e3f1251d	6725f32ba9ef07f3
214ad63d63b69ae1	214ad63d63b69ae1	555a0af07423954a

4 Automatic Preimage Attacks Framework

In this section, an automated preimage attack framework for Ascon that utilizes the linearize-and-guess approach is presented. This framework allows us to automatically trace and control the propagation of linear bits, as well as exhaustively search for the optimal structures. We successfully apply this framework to preimage attacks on 3-round Ascon in Section 5 and on 4-round Ascon in Section 6, respectively. As a result, we achieve the most advanced preimage attacks on 3-round and 4-round Ascon.

4.1 Overview

There are three stages in our attack framework, namely the optimal structures search stage, the validity of optimal structures verification stage, and the final complexity computation stage. Before overviewing the three stages, we introduce some key model parameters used in our attack framework.

The n-round preimage attack on Ascon-Xof with a 128-bit hash output is based on $(n-1)$-round optimal structure. Since the largest algebraic degree of bits within the $(n-1)$-round structure is 2^{n-2}, we refer to these 2-round (3-round) structures as *quadratic (quartic) structures* in our paper. The number of guess linear equations leaked by the hash value is defined as d_e, and the number of degrees of freedom left is defined as d_f, where $d_f \geq d_e$ is required in our attack framework. The size of random space corresponds to the number of constant bits (excluding one padding bit) in the outer part of the initial state, denoted by 2^{d_r}. Next, we overview the three stages.

Optimal Structures Search Stage. In this stage we search optimal $(n-1)$-round structure used in the n-round preimage attack by building a bit-level SAT search model. An optimal structure refers to a structure that can construct the largest linear equations leaked by the hash value while maintaining sufficient degrees of freedom, which could significantly reduce the final complexity. The details of SAT model are given in Section 4.2.

Validity of Optimal Structures Verification Stage. In Section 2.3, we have discussed a special case in the allocation model where $2^{d_r} \geq 2^{d_1}$ must hold; otherwise, the final search complexity of preimage attack will be greater than 2^{d_2}. Therefore, after the SAT solver returns an optimal structure, we should verify whether the model parameters of this optimal structure satisfy $2^{d_r} \geq 2^{d_1}$. The basic verification procedure goes as follows.

1. Calculate d_1, where d_1 equals to the total number of restricted conditions in the inner part of the initial state.

2. Calculate d_r, where d_r equals to the number of constant bits in the outer part (excluding the padding bit) and $d_r = r - 1 - d_f$ in our attack.

3. Check whether the optimal structure satisfies $2^{d_r} \geq 2^{d_1}$.

   – If $2^{d_r} \geq 2^{d_1}$, go to next stage.

   – If $2^{d_r} < 2^{d_1}$, go back to the optimal structures search stage and re-search suboptimal structures and then repeat step $1-3$.

Final Complexity Computation Stage. After getting the best valid structure, in this stage, we aim to compute the final search complexity 2^{d_2} according to the model parameters d_e, d_f. We first calculate the final gain in preimage search, denoted by $2^{0.585d_e}$. Thus, the final search complexity is calculated as $2^{d_2} = 2^{128-0.585d_e}$.

4.2 SAT Model for Searching Optimal Structure

With the key model parameters in mind, the main goal of the optimal structures search stage becomes clear. In our structure search, we aim to search for $(n-1)$-round optimal structures (see Figure 5). Initially, we construct a bit-level SAT model to depict the propagation of linear bits through the round function, which is represented as $p_L \circ p_S$ (we omit p_C as it does not affect our model). Our objective function is defined as maximizing d_e under the constraint $d_f \geq d_e$. Specially, we leverage the rotational symmetry of Ascon to accelerate the solving of our SAT model. Afterwards, we input such SAT model and objective function into the `CaDiCaL` solver to determine the existence of the $(n-1)$-round structure with d_e not less than T. If this prediction is satisfiable, we return all feasible solutions; otherwise we update the value of T with $T-1$ and ask the SAT solver to verify the satisfiability. This procedure is terminated until we get the optimal structures.

$$\begin{tikzcd} A^0 \arrow[r, "p_S"] & B^0 \arrow[r, "p_L"] & A^1 \arrow[r, "p_S"] & B^1 \arrow[r, "p_L"] & \ldots A^{n-1} \arrow[r, "p_S"] & B^{n-1} & H \arrow[l, "p_L^{-1}"'] \end{tikzcd}$$

Figure 5: An $(n-1)$-round structure used in the n-round preimage attack framework of Ascon-Xof.

Modeling the Initialization. We prepare an initial state A^0 with 320 model variables, denoted by $A^0_{y,x}$. If $A^0[y][x]$ is a linear bit, $A^0_{y,x} = 1$; otherwise $A^0_{y,x} = 0$, *i.e.*, a constant bit. Since the last c bits of the initial state and the last bit of the outer part (padding bit) are all fixed constants, we initialize their corresponding model variables as 0.

We replace 'c' with '0' and 'v' with '1' in Table 2 and then get a new Table 5. Thanks to the logic module—POSform for SymPy[5] allows us conveniently to translate this table in CNF. By extracting 5 rows (minterms) from this table as arguments to the POSform() function, 11 clauses can be obtained. The POSform() function employs simplified pairs and a redundant group elimination algorithm to convert the list of all input combinations that generate the minterms into the smallest Product-Of-Sums (POS) form. For more details please refer to https://docs.sympy.org/latest/modules/logic.html.

Table 5: Diffusion patterns in the first p_S. B^0: the output state of the first p_S.

$A^0_{0,x}$	$A^0_{1,x}$	$A^0_{2,x}$	$A^0_{3,x}$	$A^0_{4,x}$	$B^0_{0,x}$	$B^0_{1,x}$	$B^0_{2,x}$	$B^0_{3,x}$	$B^0_{4,x}$
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	1	0
1	0	0	0	0	1	1	0	0	0
1	0	0	0	0	0	1	0	1	1
1	0	0	0	0	0	1	0	0	1

Modeling the Substitution Layer p_S. For the substitution layers in subsequent rounds, we introduce 2-bit encoding method. Each bit is represented by two 0-1 variables (w_0, w_1), where $(0,1)$ stands for constant bit, $(1,1)$ stands for linear bit, and $(0,0)$ stands for unknown bit. Table 3 and Table 14 conclude the diffusion patterns in the second and third p_S, respectively. Then, we can build the SAT model for the corresponding substitution layer by translating these tables into clauses.

In order to provide a comprehensive illustration of the process of modeling the substitution layer, let's consider a SAT model for searching quadratic structures. In the first p_S operation, we select 5 rows from Table 5 and generate 11 clauses by invoking the

[5]Logic Friday, a freeware tool for Boolean logic analysis, allows us to get the same clauses but only supports Windows.

POSform() function. Moving to the second p_S, due to the independent diffusion of linear bits within each row of B^0 by the subsequent linear diffusion layer p_L, the constant row remains unchanged within the output state of p_L. This signifies that all 64 bits of $A^1[2][\star]$ are constant bits. As indicated in Table 3, there are 16 possible patterns for the quadratic structure. By using $(0, 1)$, $(1, 1)$, and $(0, 0)$ to replace 'c', 'v', and 'q', respectively, we can obtain 16 clauses from the output of the POSform() function.

Modeling the Linear Layer p_L. Let $B^i, 0 \le i < n-1$ be the output state of the i-th p_S. For the first p_L, $A^1_{y,x} = 1$ if any of $B^0[y][x], B^0[y][x-r_0], B^0[y][x-r_1]$ is a linear bit; otherwise $A^1_{y,x} = 0$. For the linear diffusion layer p_L in subsequent rounds, $A^{i+1}_{y,x} = (0, 0)$ if any of $B^i[y][x], B^i[y][x-r_0], B^i[y][x-r_1]$ is an unknown bit; else $A^{i+1}_{y,x} = (1, 1)$, if any of them is a linear bit; otherwise $A^{i+1}_{y,x} = (0, 1)$.

Modeling the Matching Hash Bit. At the model finalization, we introduce 320 model variables $A^{n-1}_{y,x}$ to represent the last state A^{n-1} of structure. If $A^{n-1}[y][x]$ is not an unknown bit, $A^{n-1}_{y,x} = 1$; otherwise $A^{n-1}_{y,x} = 0$. We exploit the Observation 3 to count the number of matching hash bits. For each column, if $A^{n-1}[0][i], A^{n-1}[2][i], A^{n-1}[3][i]$ $(0 \le i < 64)$, are linear bits or constant bits, we say there is a 1-bit hash matching, and one guess linear equation is constructed with a probability of $\frac{3}{4}$ as listed in Equation 4. Additionally, we introduce 64 variables, denoted by $E_i, 0 \le i < 64$ for $A^{n-1}[\star][i]$ to indicate which column matches successfully, if $E_i = 1$ means there is a 1-bit hash matching; otherwise, $E_i = 0$.

$$A^{n-1}[0][i] + A^{n-1}[2][i] + A^{n-1}[3][i] = B^{n-1}[0][i] \tag{4}$$

All possible cases of matching can be enumerated in Table 6, and if we put all of them into POSform() function, 4 clauses are generated.

Table 6: Possible cases of matching at the model finalization.

$A^{n-1}_{0,x}$	$A^{n-1}_{1,x}$	$A^{n-1}_{2,x}$	$A^{n-1}_{3,x}$	$A^{n-1}_{4,x}$	E_x	$A^{n-1}_{0,x}$	$A^{n-1}_{1,x}$	$A^{n-1}_{2,x}$	$A^{n-1}_{3,x}$	$A^{n-1}_{4,x}$	E_x
0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	1	0	0	0	1	0
0	1	0	0	0	0	0	1	0	0	1	0
1	1	0	0	0	0	1	1	0	0	1	0
0	0	1	0	0	0	0	0	1	0	1	0
1	0	1	0	0	0	1	0	1	0	1	0
0	1	1	0	0	0	0	1	1	0	1	0
1	1	1	0	0	0	1	1	1	0	1	0
0	0	0	1	0	0	0	0	0	1	1	0
1	0	0	1	0	0	1	0	0	1	1	0
0	1	0	1	0	0	0	1	0	1	1	0
1	1	0	1	0	0	1	1	0	1	1	0
0	0	1	1	0	0	0	0	1	1	1	0
1	0	1	1	0	1	1	0	1	1	1	1
0	1	1	1	0	0	0	1	1	1	1	0
1	1	1	1	0	1	1	1	1	1	1	1

Objective Function. The objective function is to maximize d_e under the constraint $d_f \ge d_e$. That is:

$$\begin{cases} d_f = \sum A^0_{0,x} \\ Maximize: \quad d_e = \sum E_x \end{cases} \tag{5}$$

Generally, encoding a limit on the maximum number of active variables in a given set is commonly referred to as a cardinality constraint encoding. The modulo totalizer for k-cardinality encoding (kmtotalizer for short) [MIM14] shows well with the least number of clauses and variables. We employ the kmtotalizer encoding method to describe above cardinality constraint.

Rotational Symmetry. We find that the searched structures for Ascon exhibit rotation invariance, which means rotating each word of a specific structure by a fixed offset will result in another valid structure with equivalent properties. This rotational symmetry provides a significant advantage by reducing the number of structures that need to be considered in our SAT model. Therefore, during the search phase for all feasible solutions, once we obtain a solution, we will incorporate the concept of rotational symmetry by adding 64 additional clauses to the original SAT problem, which can effectively restrict the acquired solutions from the solution space. By doing so, the solver will exclude rotationally-equivalent solutions and continue searching for new ones that are distinct.

5 Improved Preimage Attacks on 3-round Ascon-Xof

In this section, we present a 3-round preimage attack on Ascon-Xof with a 128-bit hash output. Specially, our attack framework utilizes multiple structures rather than relying on a single 2-round structure (quadratic structure). Initially, an optimal quadratic structure is identified with the model parameters $d_f = d_e = 27$, $d_r = 36$, $d_1 = 45$. However, the condition $2^{d_r} \geq 2^{d_1}$ is not satisfied within this structure. To overcome this limitation, the strategy of combining multiple structures is proposed. We exhaustively search for all quadratic structures with the same model parameters $d_f = 27, d_e = 27, d_r = 36$, effectively reducing the restricted conditions from $d_1 = 45$ to $d_1 < 36$. Consequently, the final complexity of the preimage attack on 3-round Ascon-Xof is significantly reduced to $d_2 = 2^{112.205}$.

5.1 Quadratic Structures

We apply our SAT model to search for all quadratic structures with d_e greater than 26. Throughout the search, we identify the optimal quadratic structure leaves 27 degrees of freedom and 27 guess linear equations leaked by the hash value hold with a probability of $(3/4)^{27} \approx 2^{-11.205}$. The SAT solver `CaDiCaL` successfully returned all 149 solutions in approximately 17 hours (see Table 7). Finally, we obtain $149 \times 37 = 5513$ [6] quadratic structures under the same model parameters $d_f = 27$, $d_e = 27$, $d_r = 36$.

Table 7: Summary of quadratic structures. N: the number of searched solutions (considering rotational symmetry). UNSAT: no such structure exists. $\surd$: these structures are used in our 3-round attack framework.

d_f	d_e	d_r	n_0	D_1	N	Note
≥ 28	≥ 28	-	-	-	0	UNSAT
27	27	36	≥ 10	-	0	UNSAT
27	27	36	9	45	2	$\surd$
27	27	36	8	46	147	$\surd$

Calculating the Number of Conditions for Each Single Quadratic Structure. Let D_1 be the number of conditions determined by the inner part of current structure, which is correlated to the number of '1's in $A^0_{0,x}$. In each column of the initial state, if $A^0_{0,x} = 1$, two conditions determined by the inner part must be satisfied (see Table 2), and if the third output bit $B^0[3][x]$ is a linear bit, denoted as $B^0_{3,x} = 1$, it implies that only one

[6]Each solution actually corresponds to 64 structures due to the symmetry property. However, only $64 - 27 = 37$ of 64 structures are valid as the last bit is a padding bit (cannot be a linear bit).

condition determined by the inner part must be satisfied — the structure still holds when $B^0_{3,x} = 0$. For example, as shown in Table 2, if one column belongs to case (2), then only one linear condition $a_1 = 0$ is imposed on this column. One of 5513 quadratic structures with 46 conditions are presented in Figure 6.

We summarize these relationships as follows:

$$\begin{cases} D_1 = 2d_f - n_0 \\ n_0 = \sum B^0_{3,x} \end{cases} \tag{6}$$

5.2 The Union of Multiple Subspaces

In Section 3.1, we discuss our attack framework for Ascon-Xof. In each single 2-round quadratic structure, the initial state contains 36 free constant bits (excluding the padding bit) and $D_1 = 45$ or $D_1 = 46$ linear conditions determined by the inner part (capacity). Thus $d_r = 36$ and $d_1 \geq 45$, dissatisfying $2^{d_1} \leq 2^{d_r}$. To satisfy the restriction, we raise a strategy that makes use of multiple (similar) quadratic structures — as long as the conditions of any single one among multiple quadratic structures are satisfied, then attackers can conduct the preimage attack through the specific structure. Under this strategy, d_1 can be effectively decreased and in the following, we will prove $2^{d_1} < 2^{d_r} = 2^{36}$.

Let S denote the entire capacity space ($|S| = 2^{256}$), and S_i denote the subspace of S that satisfies d_1 conditions of the i-th structure ($i = 1, 2, \ldots, 5513$). Then the probability of a random capacity belonging to any subspace (satisfying corresponding d_1 conditions) is:

$$P = \frac{1}{d_1} = \frac{|S_1 \cup S_2 \cup \cdots \cup S_{5513}|}{|S|} \tag{7}$$

Thus we need to estimate the union of multiple subspaces. According to the *inclusion-exclusion counting principle*, it holds that:

$$\begin{aligned} \left|\bigcup_{i=1}^{n} S_i\right| &= \sum_{1\leq i\leq n} |S_i| - \sum_{1\leq i<j\leq n} |S_i \cap S_j| + \sum_{1\leq i<j<k\leq n} |S_i \cap S_j \cap S_k| \\ &\quad - \sum_{1\leq i<j<k<l\leq n} |S_i \cap S_j \cap S_k \cap S_l| + \ldots \\ &\geq \sum_{1\leq i\leq n} |S_i| - \sum_{1\leq i<j\leq n} |S_i \cap S_j| \end{aligned} \tag{8}$$

Since the first two terms are both easy to calculate, we can use this formula to estimate the union and our target is $\sum |S_i| - \sum |S_i \cap S_j| > 2^{-36} \times |S|$. For convenience, we always regard $|S_i| = 2^{-46} \times |S|$ even though the number of conditions may be 45. As for $|S_i \cap S_j|$, this can be derived by the number of common conditions between the i-th and j-th 2-round quadratic structures — more specifically, t (out of 46) common conditions infers $|S_i \cap S_j| = 2^{t-92} \times |S|$. A special case should be noticed that two quadratic structures may contain contrary conditions (*e.g.* $A^0_{1,x} = 0$ and $A^0_{1,x} = 1$). Under this case, $S_i \cap S_j = \emptyset$ no matter what other conditions are.

According to the principles above, we can obtain a basic estimation about the union of 5513 subspaces. The results are given in Table 8.

However, if we directly use above results to calculate $\sum |S_i| - \sum |S_i \cap S_j|$, we can even obtain a negative value ($5513 \times 2^{-46} - 3330 \times 2^{45-92} - 6732 \times 2^{44-92} - 21285 \times 2^{43-92} < 0$). That's because two quadratic structures may be quite similar and thus contain a large number of common conditions. For example, suppose (S_1, S_2) and (S_1, S_3) both contain 45 common conditions, which results in $|S_1| - \sum |S_1 \cap S_j| < |S_1| - |S_1 \cap S_2| - |S_1 \cap S_3| = 0$. Then under this case, removing S_1 will increase the calculated result of estimation formula

Table 8: Statistics of common conditions ($\binom{5513}{2} = 15193828$ combinations of $S_i \cap S_j$ in total). Com.Cons: the number of common conditions. Sub.Comb: the number of subspace combinations.

Com.Cons	Sub.Comb	Com.Cons	Sub.Comb	Com.Cons	Sub.Comb
-1†	11091422	34	175468	40	60438
...	...	35	191705	41	44213
30	104925	36	174250	42	15924
31	122350	37	149620	43	21285
32	141769	38	126097	44	6732
33	167165	39	80648	45	3330

†-1 indicates contrary conditions exist.

Algorithm 1 Greedy Subspace Filter

Require: $n = 5513$.
Ensure: The leaving subspaces.

```
1: $n = 5513$;
2: Calculate the $n \times n$ array of $|S_i \cap S_j|$ (fill the diagonal by 0);
3: Initialize $filtertag$ = true;
4: while $filtertag$ do
5:    Initialize $maxcolumn$ and $maxcolumnsum$;
6:    for $i = 1 : n$ do
7:       Calculate the $i$-th column sum of the array;
8:       Update $maxcolumn$ and $maxcolumnsum$;
9:    end for
10:   if $maxcolumnsum > |S_{maxcolumn}|$ then
11:      Remove subspace $S_{maxcolumn}$;
12:      Update the array entries related to $S_{maxcolumn}$ by filling 0;
13:   else
14:      $filtertag$ = false;
15:   end if
16: end while
```

$\sum |S_i| - \sum |S_i \cap S_j|$. This example inspires us to design a filter to select proper subspaces. After filtering, the leaving subspaces can keep a certain "distance" from each other, so that $\sum |S_i| - \sum |S_i \cap S_j|$ would not be increased by removing any of them.

A subspace filter based on greedy algorithm is given in Algorithm 1. Finally, the filter leaves 2426 subspaces. And the statistics of common conditions are given in Table 9.

According to above results, we have:

$$\begin{aligned}&\left(\sum_{1\le i\le 2426} |S_i| - \sum_{1\le i<j\le 2426} |S_i \cap S_j|\right) / |S| \\ &= 2426 \times 2^{-46} - 502 \times 2^{44-92} - 2173 \times 2^{43-92} - \ldots - 12367 \times 2^{28-92} - \ldots \\ &\approx 1529 \times 2^{-46} > 2^{-36}\end{aligned} \tag{9}$$

Thus, $2^{d_1} < 2^{d_r} = 2^{36}$ is proved.

Table 9: Statistics of common conditions ($\binom{2426}{2} = 2941525$ combinations of $S_i \cap S_j$ in total). Com.Cons: the number of common conditions. Sub.Comb: the number of subspace combinations.

Com.Cons	Sub.Comb	Com.Cons	Sub.Comb	Com.Cons	Sub.Comb
-1†	2281258	34	23995	40	5906
...	...	35	21878	41	3844
30	17405	36	18571	42	1842
31	20866	37	14209	43	2173
32	21923	38	10703	44	502
33	25730	39	7955	45	0

†-1 indicates contrary conditions exist.

5.3 Final Complexity

To support our theoretical analysis, we perform an experiment to verify whether we can find such initial states that satisfy 46 conditions determined by one of the 2426 quadratic structures within a data complexity of 2^{36}. It turns out that we successfully find one, which is presented in Table 10.

Table 10: The initial state that satisfies the 46 conditions of the 708-th quadratic structure.

Initial inputs	Initial outputs
`00400c0000000000`	`b57e273b814cd416`
`0000000000000000`	`2b51042562ae2420`
`0000000000000000`	`66a3a7768ddf2218`
`0000000000000000`	`5aad0a7a8153650c`
`0000000000000000`	`4f3e0e32539493b6`
Absorb M_1	M_1 outputs
`1fd8c82f71985a9a`	`6b6946ff0830a067`
`2b51042562ae2420`	`787c140a83c303d0`
`66a3a7768ddf2218`	`1323ea74c00d171c`
`5aad0a7a8153650c`	`ae41eeef1ced7aee`
`4f3e0e32539493b6`	`a794219fa35585f5`

In summary, the final search complexity of preimage attack on 3-round Ascon-Xof is $2^{128-0.585\times 27} = 2^{112.205}$ (satisfying the padding rule), which is the best known preimage attack so far.

6 Improved Preimage Attacks on 4-round Ascon-Xof

Different from the quadratic structures that only include linear conditions in the initial state, the quartic structures also include linear conditions in the intermediate state A^1 and quadratic conditions in the intermediate state A^2. These conditions are determined by the controllable constant bits of the outer part of the initial state. To handle these quadratic conditions, we leverage the degrees of freedom associated with the controllable constant bits to linearize them. This ensures that each turn we set a qualified constant of the outer part to restart this quartic structure (satisfying all linear and quadratic conditions) with a probability of 1. Furthermore, the restriction $d_1 \leq d_r < 57$ in our 4-round attack framework must be satisfied.

Our preimage attack framework on 4-round Ascon-Xof is based on a single quartic structure with the model parameters $d_f = d_e = 6$, $d_r = 24$, $d_1 = 14$, as shown in Figure 7. As a result, we present the best known preimage attack on 4-round Ascon-Xof.

6.1 Quartic Structures

Quadratic Conditions Exist in the Quartic Structures. By translating all input-output patterns presented in Table 2, Table 3 and Table 14 into CNFs, we establish a crucial substitution layer model within our quartic structure search model. However, some quadratic conditions in the intermediate state are necessarily introduced for linearizing those bits belonging to $A^3[0][\star]$, $A^3[2][\star]$, or $A^3[3][\star]$ and further getting as many as possible guess equations. Such as the following quadratic condition $A^2[1][27] = 0$ (see Figure 7) in the third substitution layer can linearize the unknown bit $B^2[0][27]$ as follows.

$$\begin{aligned} B^2[0][27] &= (A^2[4][27] + A^2[2][27] + A^2[0][27] + 1)A^2[1][27] \\ &\quad + A^2[3][27] + A^2[2][27] + A^2[0][27] \\ &= A^2[3][27] + A^2[2][27] + A^2[0][27] \end{aligned}$$

As $B^2[0][46]$, $B^2[0][18]$ as well as linearized $B^2[0][27]$ are already indeed linear bits, in this way, we successfully transform $A^3[0][46]$ into a linear bit as below.

$$A^3[0][46] = B^2[0][46] + B^2[0][27] + B^2[0][18]$$

According to our summary in Table 14, there are 4 classes of quadratic conditions. Each quadratic condition involves at least 3 quadratic terms (see Equation 10).

$$\begin{aligned} A^2[0][x] &= B^1[0][x] + B^1[0][x-19] + B^1[0][x-28] \\ A^2[1][x] &= B^1[1][x] + B^1[1][x-61] + B^1[0][x-39] \\ A^2[3][x] &= B^1[3][x] + B^1[3][x-10] + B^1[3][x-17] \\ A^2[3][x] + A^2[4][x] &= B^1[3][x] + B^1[3][x-10] + B^1[3][x-17] \\ &\quad + B^1[4][x] + B^1[4][x-7] + B^1[4][x-41] \end{aligned} \tag{10}$$

Optimal Quartic Structures. Based on our SAT model, we find the optimal quartic structures with the model parameters $d_f = 8$, $d_e = 8$, $d_1 = 21$. However, none of these structures can be used in our 4-round attack framework due to the inability to satisfy the extra 28 linear and 18 quadratic conditions simultaneously by insufficient $64 - 1 - 8 = 55$ free constant bits. Then, we attempt to search for suboptimal structures with the model parameters $d_f = 7$, $d_e = 7$, $d_1 = 18$. Unfortunately, those structures are still not applicable to our attack framework for the same reason. Thus, we exhaustively search for all quartic structures with the model parameters $d_f = 6$, $d_e = 6$, $d_1 = 14$. Table 11 provides a summary of quartic structures.

Table 11: Summary of quartic structures. n_1: the number of linear conditions in state A^1. n_2: the number of quadratic condition in state A^2. N: the number of searched solutions (considering rotational symmetry). UNSAT: no such structure exists. $\times$: the structures under such parameters cannot be used in our 4-round attack framework.

d_f	d_e	d_1	n_1	n_2	N	Note
≥ 9	≥ 9	-	-	-	-	UNSAT
8	8	21	28	18	-	$\times$
7	7	18	24	15	-	$\times$
6	6	14	20	13	1024	$\surd$

In these quartic structures with the model parameters $d_f = 6$, $d_e = 6$, $d_1 = 14$, the initial state includes 57 free constant bits (excluding the padding bit) and 14 linear conditions determined by the inner part of A^0, *i.e.*, $d_1 = 14$. Within each structure, there are 20 linear conditions and 13 quadratic conditions in the intermediate states.

To satisfy all conditions imposed on the intermediate states, as well as the restrictions $d_1 \le d_r < 57$, we propose a specific linearization strategy. With this strategy, 13 quadratic conditions are linearized by introducing 22 extra linear conditions[7]. This allows us to prove that d_r can reach 24, satisfying $2^{d_r} > 2^{d_1} = 2^{14}$.

6.2 Linearization Strategy

We show how to linearize quadrtic conditions using our linearization strategy by one example of a quartic structure with the model parameters $d_f = 6, d_e = 6, d_1 = 14$.

Duplicate Quadratic Terms. 13 quadratic conditions are listed in Figure 7. Since all quadratic conditions are intensively located on the 0-th and 1-st rows, 13 quadratic conditions may not correspond to 39 different quadratic terms. To facilitate observation, we extract all quadratic terms from 13 conditions listed in Table 12. It is found that 10 out of 39 quadratic terms overlap. Thus, we only need to linearize 29 quadratic terms.

Table 12: Statistics of quadratic terms of 13 quadratic conditions.

Quadratic conditions	no.1 Quadratic term	no.2 Quadratic term	no.3 Quadratic term
$A^2[0][30] = 1$	$B^1[0][30]$	$B^1[0][11]$	$B^1[0][2]$
$A^2[0][32] = 1$	$B^1[0][32]$	$B^1[0][13]$	$B^1[0][4]$
$A^2[0][57] = 1$	$B^1[0][57]$	$B^1[0][38]$	$B^1[0][29]$
$A^2[1][13] = 1$	$B^1[1][13]$	$B^1[1][16]$	$B^1[1][38]$
$A^2[1][27] = 0$	$B^1[1][27]$	$B^1[1][30]$	$B^1[1][52]$
$A^2[1][30] = 0$	$B^1[1][30]$	$B^1[1][33]$	$B^1[1][55]$
$A^2[1][32] = 1$	$B^1[1][32]$	$B^1[1][35]$	$B^1[1][57]$
$A^2[1][35] = 0$	$B^1[1][35]$	$B^1[1][38]$	$B^1[1][60]$
$A^2[1][38] = 1$	$B^1[1][38]$	$B^1[1][41]$	$B^1[1][63]$
$A^2[1][54] = 0$	$B^1[1][54]$	$B^1[1][57]$	$B^1[1][15]$
$A^2[1][57] = 1$	$B^1[1][57]$	$B^1[1][60]$	$B^1[1][18]$
$A^2[1][60] = 0$	$B^1[1][60]$	$B^1[1][63]$	$B^1[1][21]$
$A^2[1][63] = 1$	$B^1[1][63]$	$B^1[1][2]$	$B^1[1][24]$

Linearization of Quadratic Terms. According to Equation 1, quadratic terms in the 0-th and 1-st rows are generated through the formulas below (see Equation 11). As previously noted, $A^1[2][x]$ is a constant bit determined by the inner part (see Section 4.2).

$$\begin{aligned} B^1[0][x] &\leftarrow (A^1[0][x] + A^1[2][x] + A^1[4][x] + 1)A^1[1][x] \\ B^1[1][x] &\leftarrow (A^1[1][x] + A^1[3][x])A^1[2][x] + A^1[1][x]A^1[3][x] \end{aligned} \tag{11}$$

Both $B^1[0][x]$ and $B^1[1][x]$ can be linearized by introducing an extra linear condition, namely, $A^1[1][x] = \text{const}$. If $B^1[0][x]$ and $B^1[1][x]$ belong to the same column, these two terms can be linearized by a single linear condition. The linearization details are provided in Table 13. A total of 29 quadratic terms are distributed across 23 columns, with 6 columns containing 2 quadratic terms each. Moreover, one of the quadratic terms has

[7]According to our linearization strategy, we check all 1024 structures, and we find that the least number of extra conditions introduced is 22.

already been linearized through linear conditions. Thus, to linearize the remaining 29 quadratic terms, only $29 - 6 - 1 = 22$ extra linear conditions are needed. We introduce 22 variables, denoted by c_i ($1 \leq i \leq 22$), to represent the value of extra linear conditions.

Table 13: Statistics of extra linear conditions introduced to linearize all quadratic terms.

Num	Qua.terms	Linear.cons	Num	Qua.terms	Linear.cons
1	$B^1[0][2], B^1[1][2]$	$A^1[1][2] = c_1$	12	$B^1[0][32], B^1[1][32]$	$A^1[1][32] = c_{12}$
2	$B^1[0][4]$	$A^1[1][4] = c_2$	13	$B^1[1][33]$	$A^1[1][33] = c_{13}$
†	$B^1[0][11]$	$A^1[1][11] = 1$	14	$B^1[1][35]$	$A^1[1][35] = c_{14}$
3	$B^1[0][13], B^1[1][13]$	$A^1[1][13] = c_3$	15	$B^1[0][38], B^1[1][38]$	$A^1[1][38] = c_{15}$
4	$B^1[1][15]$	$A^1[1][15] = c_4$	16	$B^1[1][41]$	$A^1[1][41] = c_{16}$
5	$B^1[1][16]$	$A^1[1][16] = c_5$	17	$B^1[1][52]$	$A^1[1][52] = c_{17}$
6	$B^1[1][18]$	$A^1[1][18] = c_6$	18	$B^1[1][54]$	$A^1[1][54] = c_{18}$
7	$B^1[1][21]$	$A^1[1][21] = c_7$	19	$B^1[1][55]$	$A^1[1][55] = c_{19}$
8	$B^1[1][24]$	$A^1[1][24] = c_8$	20	$B^1[0][57], B^1[1][57]$	$A^1[1][57] = c_{20}$
9	$B^1[1][27]$	$A^1[1][27] = c_9$	21	$B^1[1][60]$	$A^1[1][60] = c_{21}$
10	$B^1[0][29]$	$A^1[1][29] = c_{10}$	22	$B^1[1][63]$	$A^1[1][63] = c_{22}$
11	$B^1[0][30], B^1[1][30]$	$A^1[1][30] = c_{11}$			

†$A^1[1][11] = 1$ coincides with an existing linear condition.

Calculating the Random Space 2^{d_r}. With this specific linearization strategy, we obtain 55 linear equations in total, which consist of 20 original linear conditions, 22 extra linear conditions, and 13 quadratic conditions that have been linearized. It is important to note that the linearization strategy does not compress the size of the random space, as each extra equation can be assigned a value of either 0 or 1. Consequently, $2^{d_r} = 2^{57-20-13} = 2^{24}$, satisfying the condition $2^{d_r} \geq 2^{d_1}$.

The Complexity of Solving Time. According to the number of bit operations in *Gaussian Emilination*, where n is the total number of variables (*i.e.*, free constant bits here), the calculated result is

$$\begin{aligned}&(n-1)\times n + (n-2)\times(n-1) + \ldots + 1 \times 2\\ &= \frac{1}{3}[n^3 - (n-1)^3 - 1] + \frac{1}{3}[(n-1)^3 - (n-2)^3 - 1] + \ldots + \frac{1}{3}[2^3 - 1^3 - 1]\\ &= \frac{1}{3}(n^3 - n)\end{aligned} \tag{12}$$

Moreover, we can reduce the number of bit operations by precalculating 44 rows and updating only 13 rows in each turn[8]. After precalculating, the calculation becomes:

$$[(57 + 56 + \ldots + 14) + (57 + 56 + \cdots + 13) + \ldots + (57 + 56 + \ldots + 2)] = 21304$$

In comparison, the total of bit operations[9] for a 4-round Ascon is $4 \times 2688 = 10752$. Regarding the linear system, it is composed of 55 equations in 57 variables. During each turn of Gaussian Elimination, it is expected to obtain 4 solutions. For each solution, we utilize the quartic structure and generate 6 guess linear equations leaked by the hash. The complexity of solving time can be calculated by taking the ratio of the number of bit operations required for one Gaussian Elimination turn to the total number of bit

[8]Each turn those c_i values from Table 11 are updated, with no impacts on 20 original linear conditions and 22 extra conditions. However, it does modify 13 quadratic conditions.

[9]These 2688 bit operations for each round includes $64 \times 22 = 1408$ XOR operations of the substitution layer, $2 \times 320 = 640$ XOR operations of linear diffusion layer, and $2 \times 320 = 640$ rotation operations.

operations in 4 Ascon turns, *i.e.*, $\frac{21304}{4}/10752 \approx 2^{-1.01}$. As for the complexity of solving 6 guess linear equations, this part can be neglected since the number of bit operations is only $\frac{1}{3} \times (6^3 - 6) = 70$.

Final Complexity. In summary, by using a single quartic structure and applying a specific linearization strategy, we improve the preimage attack on 4-round Ascon-Xof with the final complexity of $2^{128-6\times 0.585} = 2^{124.49}$.

7 Conclusion

In this paper, we develop a SAT-based automatic preimage attack framework via a linearize-and-guess approach on Ascon. The main idea is to divide the preimage attack procedure into three main stages:

- Optimal Structures Search Stage. Based on in-depth analysis of the structural and algebraic properties of Ascon permutation, we construct a SAT model via linearize-and-guess approach to search for optimal structures that partially linearize the internal state of ASCON, resulting in most probabilistic hash-matching equations with sufficient degrees of freedom left.
- Validity of Optimal Structures Verification Stage. In this procedure, we propose a series of techniques as auxiliary tools, such as using the union of multiple structures, a generic greedy filter algorithm, and a specific linearization strategy for linearizing quadratic conditions, to verify the validity of the current preimage attack under the optimal structures, namely whether it satisfies the restriction $2^{d_r} \geq 2^{d_1}$.
- Final Complexity Computation Stage. In this stage, we aim to compute the final search complexity 2^{d_2} according to the model parameters d_e, *i.e.*, $2^{d_2} = 2^{128-0.585d_e}$.

Under this SAT-based automatic preimage attack framework, we improved the preimage attacks on Ascon-Xof that are reduced to 3 and 4 rounds. It is noted that our attack is still far from threatening the security of full-round Ascon.

Acknowledgments

We are grateful to the anonymous reviewers and our shepherd Siwei Sun for their valuable feedback and comments that improved the quality of the paper. This research is supported by the National Natural Science Foundation of China under (Grants No.61972249), Nanyang Technological University in Singapore under Start-up Grant 04INS000397C230, and Ministry of Education in Singapore under Grants RG91/20, the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative, the State Scholarship Fund (No.202106230206) organized by China Scholarship Council. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

References

[Arm19] Armin Biere. CADICAL at the SAT Race 2019. 2019. https://github.com/arminbiere/cadical.

[BDG+21] Zhenzhen Bao, Xiaoyang Dong, Jian Guo, Zheng Li, Danping Shi, Siwei Sun, and Xiaoyun Wang. Automatic search of meet-in-the-middle preimage attacks on AES-like hashing. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 771–804. Springer, Heidelberg, October 2021.

[BDPA13] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Keccak. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 313–314. Springer, Heidelberg, May 2013.

[BGST22] Zhenzhen Bao, Jian Guo, Danping Shi, and Yi Tu. Superposition meet-in-the-middle attacks: Updates on fundamental security of AES-like hashing. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part I*, volume 13507 of *LNCS*, pages 64–93. Springer, Heidelberg, August 2022.

[DEMS19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Preliminary analysis of Ascon-Xof and Ascon-Hash (version 0.1), 2019.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight Authenticated Encryption and Hashing. *J. Cryptol.*, 34(3):33, 2021.

[GLS16] Jian Guo, Meicheng Liu, and Ling Song. Linear structures: Applications to cryptanalysis of round-reduced Keccak. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 249–274. Springer, Heidelberg, December 2016.

[GLST22] Jian Guo, Guozhen Liu, Ling Song, and Yi Tu. Exploring SAT for cryptanalysis: (quantum) collision attacks against 6-round SHA-3. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part III*, volume 13793 of *LNCS*, pages 645–674. Springer, Heidelberg, December 2022.

[HLY21] Le He, Xiaoen Lin, and Hongbo Yu. Improved preimage attacks on 4-round Keccak-224/256. *IACR Trans. Symm. Cryptol.*, 2021(1):217–238, 2021.

[HLY22] Le He, Xiaoen Lin, and Hongbo Yu. Improved preimage attacks on round-reduced keccak-384/512 via restricted linear structures. Cryptology ePrint Archive, Report 2022/788, 2022. https://eprint.iacr.org/2022/788.

[LHY21] Xiaoen Lin, Le He, and Hongbo Yu. Improved preimage attacks on 3-round Keccak-224/256. *IACR Trans. Symm. Cryptol.*, 2021(3):84–101, 2021.

[LHY23] Xiaoen Lin, Le He, and Hongbo Yu. Practical preimage attack on 3-round keccak-256. Cryptology ePrint Archive, Report 2023/101, 2023. https://eprint.iacr.org/2023/101.

[LIMY20] Fukang Liu, Takanori Isobe, Willi Meier, and Zhonghao Yang. Algebraic Attacks on Round-Reduced Keccak/Xoodoo. Cryptology ePrint Archive, Report 2020/346, 2020. https://eprint.iacr.org/2020/346.

[LS19] Ting Li and Yao Sun. Preimage attacks on round-reduced Keccak-224/256 via an allocating approach. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 556–584. Springer, Heidelberg, May 2019.

[LSLW17] Ting Li, Yao Sun, Maodong Liao, and Dingkang Wang. Preimage attacks on the round-reduced Keccak with cross-linear structures. *IACR Trans. Symm. Cryptol.*, 2017(4):39–57, 2017.

[MIM14] António Morgado, Alexey Ignatiev, and João Marques-Silva. MSCG: robust core-guided maxsat solving. *J. Satisf. Boolean Model. Comput.*, 9(1):129–134, 2014.

[MLM21] João Marques-Silva, Inês Lynce, and Sharad Malik. Conflict-Driven Clause Learning SAT Solvers. In *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 133–182. IOS Press, Ohmsha, 2021.

[MP13] Nicky Mouha and Bart Preneel. Towards finding optimal differential characteristics for ARX: Application to Salsa20. *IACR Cryptol.*, 2013:328, 2013. https://eprint.iacr.org/2013/328.

[MS13] Pawel Morawiecki and Marian Srebrny. A SAT-based preimage analysis of reduced Keccak hash functions. *Inf. Process. Lett.*, 113(10-11):392–397, 2013.

[MWGP11] Nicky Mouha, Qingju Wang, Dawu Gu, and Bart Preneel. Differential and Linear Cryptanalysis Using Mixed-Integer Linear Programming. In *Information Security and Cryptology - 7th International Conference, Inscrypt 2011*, volume 7537, pages 57–76, Berlin, Heidelberg, 2011. Springer.

[QHD$^+$23] Lingyue Qin, Jialiang Hua, Xiaoyang Dong, Hailun Yan, and Xiaoyun Wang. Meet-in-the-Middle Preimage Attacks on Sponge-Based Hashing. In *EUROCRYPT (4)*, volume 14007 of *Lecture Notes in Computer Science*, pages 158–188. Springer, 2023.

[QZH$^+$23] Lingyue Qin, Boxin Zhao, Jialiang Hua, Xiaoyang Dong, and Xiaoyun Wang. Weak-Diffusion Structure: Meet-in-the-Middle Attacks on Sponge-based Hashing Revisited. Cryptology ePrint Archive, Paper 2023/518, 2023. https://eprint.iacr.org/2023/518.

[Raj19] Mahesh Sreekumar Rajasree. Cryptanalysis of round-reduced KECCAK using non-linear structures. In Feng Hao, Sushmita Ruj, and Sourav Sen Gupta, editors, *INDOCRYPT 2019*, volume 11898 of *LNCS*, pages 175–192. Springer, Heidelberg, December 2019.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. In *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584, pages 244–257, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[SWW18] Ling Sun, Wei Wang, and Meiqin Wang. More accurate differential properties of LED64 and Midori64. *IACR Trans. Symm. Cryptol.*, 2018(3):93–123, 2018.

[SWW21] Ling Sun, Wei Wang, and Meiqin Wang. Accelerating the search of differential and linear characteristics with the SAT method. *IACR Trans. Symm. Cryptol.*, 2021(1):269–315, 2021.

A Appendix

A.1 Valid Inputs/Outputs Pattern through the Substitution Layer p_S

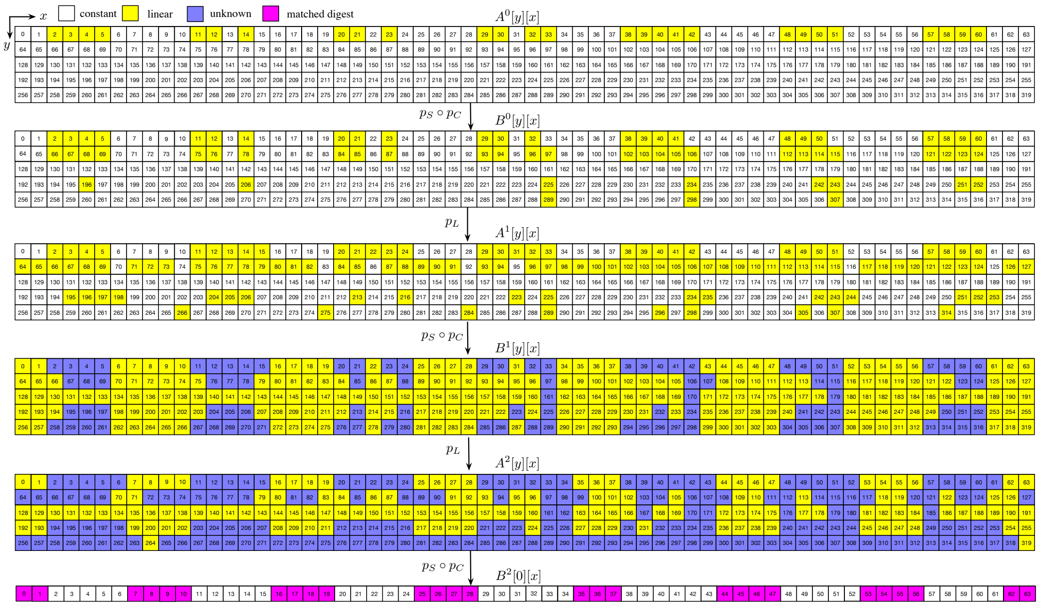
We summarize the diffusion patterns in the third p_S in Table 14 (only the first and last bits of the input can be ‘q’).

Table 14: Diffusion patterns in the third p_S. # C: represents the number of required conditions for each diffusion pattern.

Inputs	Outputs	Condition	$\#C$	Inputs	Outputs	Condition	$\#C$
ccccc	vvvvv	-	0	vcccc	vvvvv	-	0
cvccc	vvvvv	-	0	cccvc	vvvvv	-	0
ccccv	vvvvv	-	0	vvccc	qvvvq	-	0
cvccv	qvvvq	-	0	vccvc	vvvqv	-	0
cvcvc	vqvvv	-	0	vvcvc	qqvqq	-	0
vcccv	vvvqv	-	0	vvccv	qvvqq	-	0
cccvv	vvqvv	-	0	vccvv	vvqqv	-	0
cvcvv	qqqvq	-	0	vvcvv	qqqqq	-	0
ccvcc	vvvvv	-	0	ccvcv	vvvvv	-	0
ccvvc	vqvvv	-	0	ccvvv	vqqvv	-	0
cvvcc	qqvvv	-	0	cvvcv	qqvvq	-	0
cvvvc	qqvvv	-	0	cvvvv	qqqvq	-	0
vcvcc	vvvvv	-	0	vcvcv	vvvqv	-	0
vcvvc	vqvqv	-	0	vcvvv	vqqqv	-	0
vvvcc	qqvvq	-	0	vvvcv	qqvqq	-	0
vvvvc	qqvqq	-	0	vvvvv	qqqqq	-	0
qvccc	qqvqq	-	0	qvccv	qqvqq	-	0
qvcvc	qqvqq	-	0	qvcvv	qqqqq	-	0
qvvcc	qqvqq	-	0	qvvcv	qqvqq	-	0
qvvvc	qqvqq	-	0	qvvvv	qqqqq	-	0
qcccc	qqvqv	$a_1 = 0$	1	qcccc	qqvvv	$a_1 = 0; a_3 + a_4 = 1$	2
qcccc	vqvqq	$a_1 = 1$	1	qcccc	vqvvq	$a_1 = 1; a_3 + a_4 = 1$	2
qcccv	qqvqv	$a_1 = 0$	1	qcccv	vqvqq	$a_1 = 1$	1
qccvc	qqvqv	$a_1 = 0$	1	qccvc	vqvqq	$a_1 = 1$	1
qccvv	qqqqv	$a_1 = 0$	1	qccvv	vqqqq	$a_1 = 1$	1
qcvcc	qqvqv	$a_1 = 0$	1	qcvcc	qqvvv	$a_1 = 0; a_3 + a_4 = 1$	2
qcvcc	vqvqq	$a_1 = 1$	1	qcvcc	vqvvq	$a_1 = 1; a_3 + a_4 = 1$	2
qcvcv	qqvqv	$a_1 = 0$	1	qcvcv	vqvqq	$a_1 = 1$	1
qcvvc	qqvqv	$a_1 = 0$	1	qcvvc	vqvqq	$a_1 = 1$	1
qcvvv	vqqqq	$a_1 = 1$	1	qvvcc	qqvvq	$a_3 + a_4 = 1$	1
qcvvv	qqqqv	$a_1 = 0$	1	qvccc	qqvvq	$a_3 + a_4 = 1$	1
ccccq	qqqqv	$a_1 = 1$	1	ccccq	qqqvv	$a_1 = 1; a_0 = 1$	2
ccccq	qqvqv	$a_1 = 1; a_3 = 1$	2	ccccq	qqvvv	$a_1 = 1; a_0 = 1; a_3 = 1$	3
ccccq	vqqqq	$a_1 = 0$	1	ccccq	vqqvq	$a_1 = 0; a_0 = 1$	2
ccccq	vqvqq	$a_1 = 0; a_3 = 1$	2	ccccq	vqvvq	$a_1 = 0; a_3 = 1; a_0 = 1$	3
cccvq	qqqqv	$a_1 = 1$	1	cccvq	qqqvv	$a_1 = 1; a_0 = 1$	2
cccvq	vqqqq	$a_1 = 0$	1	cccvq	vqqvq	$a_1 = 0; a_0 = 1$	2
ccvcq	qqqqv	$a_1 = 1$	1	ccvcq	qqqvv	$a_1 = 1; a_0 = 1$	2
ccvcq	qqvqv	$a_1 = 1; a_3 = 1$	2	ccvcq	qqvvv	$a_1 = 1; a_0 = 1; a_3 = 1$	3
ccvcq	vqqqq	$a_1 = 0$	1	ccvcq	vqqvq	$a_1 = 0; a_0 = 1$	2
ccvcq	vqvqq	$a_1 = 0; a_3 = 1$	2	ccvcq	vqvvq	$a_1 = 0; a_3 = 1; a_0 = 1$	3
ccvvq	qqqqv	$a_1 = 1$	1	ccvvq	qqqvv	$a_1 = 1; a_0 = 1$	2
ccvvq	vqqqq	$a_1 = 0$	1	ccvvq	vqqvq	$a_1 = 0; a_0 = 1$	2
cvccq	qqqvq	$a_0 = 1$	1	cvccq	qqvqq	$a_3 = 1$	1
cvccq	qqvvq	$a_0 = 1; a_3 = 1$	2	cvcvq	qqqvq	$a_0 = 1$	1
cvvcq	qqqvq	$a_0 = 1$	1	cvvcq	qqvqq	$a_3 = 1$	1
cvvcq	qqvvq	$a_3 = 1; a_0 = 1$	2	cvvvq	qqqvq	$a_0 = 1$	1
vcccq	qqqqv	$a_1 = 1$	1	vcccq	qqvqq	$a_3 = 1$	1
vcccq	qqvqv	$a_1 = 1; a_3 = 1$	2	vccvq	qqqqv	$a_1 = 1$	1
vcvcq	qqqqv	$a_1 = 1$	1	vcvcq	qqvqq	$a_3 = 1$	1
vcvcq	qqvqv	$a_1 = 1; a_3 = 1$	2	vcvvq	qqqqv	$a_1 = 1$	1
vvccq	qqvqq	$a_3 = 1$	1	vvvcq	qqvqq	$a_3 = 1$	1
cvccq	qqqqq	-	0	cvcvq	qqqqq	-	0
cvvcq	qqqqq	-	0	cvvvq	qqqqq	-	0
vcccq	qqqqq	-	0	vccvq	qqqqq	-	0
vcvcq	qqqqq	-	0	vcvvq	qqqqq	-	0
vvccq	qqqqq	-	0	vvcvq	qqqqq	-	0
vvvvq	qqqqq	-	0	qcccq	qqqqq	-	0
qccvq	qqqqq	-	0	qcvcq	qqqqq	-	0
qcvvq	qqqqq	-	0	qvccq	qqqqq	-	0
qvcvq	qqqqq	-	0	qvvcq	qqqqq	-	0
qvvvq	qqqqq	-	0	vvvcq	qqqqq	-	0

A.2 3-round Preimage Attack with Quadratic Structures

We finally obtain $149 \times 37 = 5513$ quadratic structures under the same model parameters $d_f = 27$, $d_e = 27$, $d_r = 36$ based on our SAT model. One of these structures with 46 conditions are shown in Figure 6.



conditions determined by the inner part of A^0:

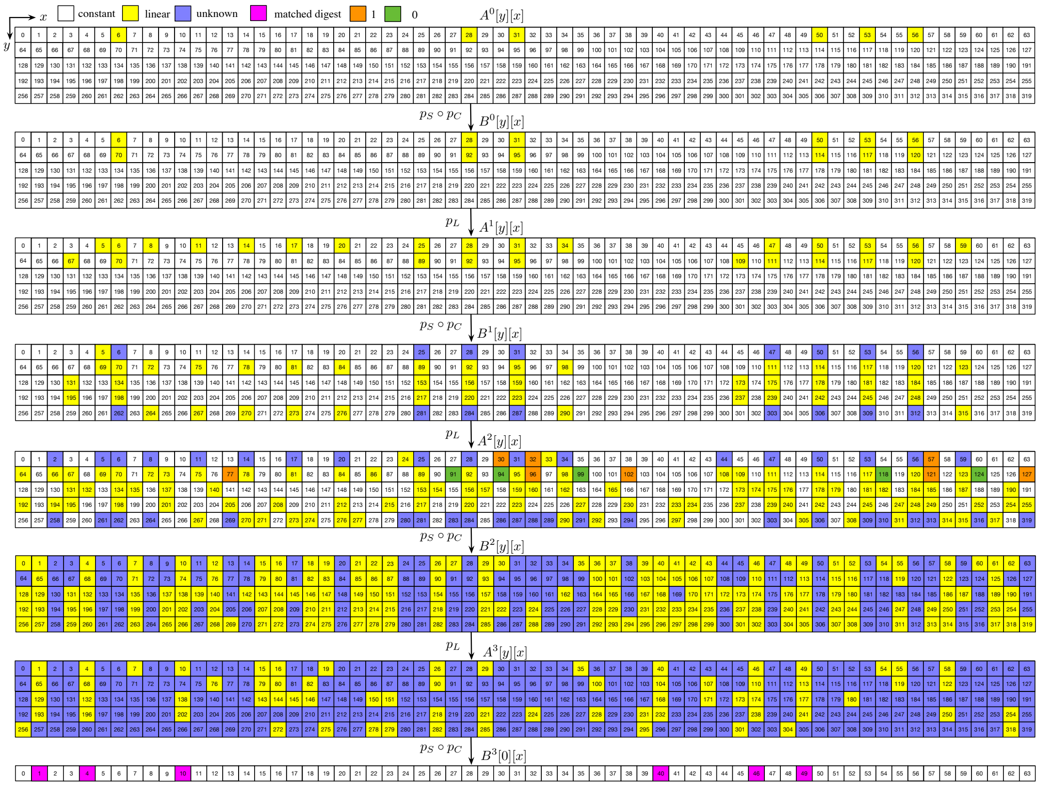
condition list:

1: A^0[1][2]=0
2: A^0[3][2]+A^0[4][2]=1
3: A^0[1][3]=0
4: A^0[3][3]+A^0[4][3]=1
5: A^0[1][4]=0
6: A^0[1][5]=0
7: A^0[3][5]+A^0[4][5]=1
8: A^0[1][11]=0
9: A^0[3][11]+A^0[4][11]=1
10: A^0[1][12]=0
11: A^0[3][12]+A^0[4][12]=1
12: A^0[1][14]=0
13: A^0[1][20]=0
14: A^0[3][20]+A^0[4][20]=1
15: A^0[1][21]=0
16: A^0[3][21]+A^0[4][21]=1
17: A^0[1][23]=0
18: A^0[3][23]+A^0[4][23]=1
19: A^0[1][29]=0
20: A^0[3][29]+A^0[4][29]=1
21: A^0[1][30]=0
22: A^0[3][30]+A^0[4][30]=1
23: A^0[1][32]=0
24: A^0[3][32]+A^0[4][32]=1
25: A^0[1][33]=1
26: A^0[1][38]=0
27: A^0[3][38]+A^0[4][38]=1
28: A^0[1][39]=0
29: A^0[3][39]+A^0[4][39]=1
30: A^0[1][40]=0
31: A^0[3][40]+A^0[4][40]=1
32: A^0[1][41]=0
33: A^0[3][41]+A^0[4][41]=1
34: A^0[1][42]=1
35: A^0[1][48]=0
36: A^0[3][48]+A^0[4][48]=1
37: A^0[1][49]=0
38: A^0[3][49]+A^0[4][49]=1
39: A^0[1][50]=0
40: A^0[1][51]=1
41: A^0[1][57]=0
42: A^0[3][57]+A^0[4][57]=1
43: A^0[1][58]=0
44: A^0[3][58]+A^0[4][58]=1
45: A^0[1][59]=0
46: A^0[1][60]=0

Figure 6: One example of quadratic structures.

A.3 4-round Preimage Attack with Quartic Structures

We list one of quartic structures under the model parameters $d_e = 27$, $d_f = 27$, $d_1 = 14$, and $d_r = 24$ based on our SAT model in Figure 7. Note that the following conditions $A^1[2][3] = 0$, $A^1[2][45] = 0$ in red are in fact determined by the inner part of A^0, so we put them uniformly in the conditions of A^0.



conditions determined by the inner part of A^0:

conditions in A^0:

1: A^0[1][6]=0
2: A^0[3][6]+A^0[4][6]=1
3: A^0[1][28]=0
4: A^0[3][28]+A^0[4][28]=1
5: A^0[1][31]=0
6: A^0[3][31]+A^0[4][31]=1
7: A^0[1][50]=0
8: A^0[3][50]+A^0[4][50]=1
9: A^0[1][53]=0
10: A^0[3][53]+A^0[4][53]=1
11: A^0[1][56]=0
12: A^0[3][56]+A^0[4][56]=1
13: A^1[2][3]=0
14: A^1[2][45]=0

conditions determined by the intermediate A^1 and A^2:

conditions in A^1:

1: A^1[3][3]=1
2: A^1[0][3]+A^1[4][3]=1
3: A^1[3][45] = 1
4: A^1[0][45]+A^1[4][45]=1
5: A^1[1][5]=0
6: A^1[3][5]+A^1[4][5]=1
7: A^1[1][8]=1
8: A^1[3][8]+A^1[4][8]=1
9: A^1[1][11]=1
10: A^1[3][11]+A^1[4][11]=1
11: A^1[1][14]=1
12: A^1[3][14]+A^1[4][14]=1
13: A^1[1][17]=1
14: A^1[3][17]+A^1[4][17]=1
15: A^1[1][20]=1
16: A^1[3][20]+A^1[4][20]=1
17: A^1[1][34]=1
18: A^1[3][34]+A^1[4][34]=1
19: A^1[1][59]=1
20: A^1[3][59]+A^1[4][59]=1

conditions in A^2:

1: A^2[0][30] = 1
2: A^2[0][32] = 1
3: A^2[0][57] = 1
4: A^2[1][13] = 1
5: A^2[1][27] = 0
6: A^2[1][30] = 0
7: A^2[1][32] = 1
8: A^2[1][35] = 0
9: A^2[1][38] = 1
10: A^2[1][54] = 0
11: A^2[1][57] = 1
12: A^2[1][60] = 0
13: A^2[1][63] = 1

Figure 7: One example of quartic structures.