Preimage Attacks on Reduced-Round Ascon-Xof[⋆]

Seungjun Baek[1], Giyoon Kim[1], and Jongsung Kim[1,2]

[1] Department of Financial Information Security, Kookmin University, Republic of Korea
`{hellosj3,gi0412,jskim}@kookmin.ac.kr`
[2] Department of Information Security, Cryptology, and Mathematics, Kookmin University, Republic of Korea

Abstract. Ascon, a family of algorithms that supports authenticated encryption and hashing, has been selected as the new standard for lightweight cryptography in the NIST Lightweight Cryptography Project. Ascon's permutation and authenticated encryption have been actively analyzed, but there are relatively few analyses on the hashing. In this paper, we concentrate on preimage attacks on Ascon-Xof. We focus on linearizing the polynomials leaked by the hash value to find its inverse. In an attack on 2-round Ascon-Xof, we carefully construct the set of guess bits using a greedy algorithm in the context of guess-and-determine. This allows us to attack Ascon-Xof more efficiently than the method in Dobraunig et al., and we fully implement our attack to demonstrate its effectiveness. We also provide the number of guess bits required to linearize one output bit after 3- and 4-round Ascon's permutation, respectively. In particular, for the first time, we connect the result for 3-round Ascon to a preimage attack on Ascon-Xof with a 64-bit output. Our attacks primarily focus on analyzing weakened versions of Ascon-Xof, where the weakening involves setting all the IV values to 0 and omitting the round constants. Although our attacks do not compromise the security of the full Ascon-Xof, they provide new insights into their security.

Keywords: NIST · Ascon· Ascon-Xof· hash function · preimage attack

1 Introduction

As the Internet of Things has grown, related fields such as sensor networks, healthcare, distributed control systems, and virtual physical systems have evolved, and the use of small devices is also increasing considerably. Small devices provide convenience and usability to users, but these advantages are meaningful only if there are no concerns regarding the security of the devices. The transition from desktop computers to small devices engenders a wide range of new security and

[⋆] This paper has been accepted by Designs, Codes and Cryptography.

privacy concerns, primarily owing to the lack of resources in small devices, which are also known as constrained devices.

The National Institute of Standards and Technology (NIST) has been working on a Lightweight Cryptography (LWC) Standardization Process to standardize one or more Authenticated Encryption with Associated Data (AEAD) and hashing schemes suitable for constrained devices [28]. NIST selected 56 first-round candidates in April 2019 and 32 second-round candidates in August 2019. In March 2021, NIST announced ten finalists for the final round of the selection process. Finally, in February 2023, NIST decided to standardize Ascon [12] through the NIST mailing list.

Ascon is not only the NIST LWC selection algorithm, but also the primary choice for lightweight applications in the final portfolio of the CAESAR competition [9]. The Ascon family supports authenticated ciphers Ascon-128, Ascon-128a, and Ascon-80pq, hash functions Ascon-Hash and Ascon-Hasha, and extendable output functions Ascon-Xof and Ascon-Xofa. All schemes commonly use 320-bit Ascon's permutation, wherein three AEAD schemes and four hashing schemes are based on duplex construction [3] and sponge construction [4], respectively. Isap [6], one of the NIST LWC finalists, also uses Ascon's permutation, which can be directly affected by Ascon analysis.

Since the CAESAR competition, the cryptography community has conducted extensive studies on Ascon, including the analysis of the underlying permutation [8, 7, 33, 19, 22, 1, 36, 18, 5, 17, 27] and Ascon AEAD [20, 8, 14, 24, 25, 26, 18, 31]. However, Ascon-Hash and Ascon-Xof have not been comprehensively studied. The designers of Ascon proposed preimage attacks on Ascon-Xof, revealing that it is possible to linearize 2-round Ascon's permutation to find preimages of round-reduced Ascon-Xof [10]. They provided noteworthy insights into the attack, but the attack process and complexity analysis were not covered in detail. In addition, they noted that an upper bound on the degree of the round-reduced Ascon's permutation can be used to marginally speed up a brute-force search for preimages by using a large amount of memory, as suggested by Bernstein [2]. Zong et al. [37] proposed a collision attack on 2-round Ascon-Hash and Ascon-Xof, and the latter could be attacked within a practical amount of time. Gerault et al. [18] improved the collision attack on 2-round Ascon-Hash by using a differential trail with a higher probability than the existing one found based on Constraint Programming. Qin et al. [29, 30] proposed preimage attacks on 3- and 4-round Ascon-Xof with a 128-bit output using a meet-in-the-middle approach. Very recently, Li et al. [23] improved Qin et al.'s results by employing a SAT-based automatic preimage attack framework that uses a linearize-and-guess approach.

An extendable output function (XOF) is a function on bit strings (also called messages) in which the output can be extended to any desired hash length [16]. The XOF can be seen as a generalization of hash functions where the output length is not fixed, but is potentially infinite. This property allows anyone who wants to output a long- or short-length hash, including a 256-bit hash, using

XOF. XOF functions can be used as key derivation functions, stream ciphers, and mask generation functions for RSA-OAEP style padding.

SHAKE128 and SHAKE256 [15] are the first XOFs that NIST has standardized, and there is no standardized XOF thereafter. As ASCON is selected as the final algorithm of NIST LWC, it is highly likely that ASCON-XOF will be officially standardized as XOF. Weatherley [34] proposed and implemented a mode using ASCON-XOF as a primitive of an algorithm such as KMAC [21]. In addition, cSHAKE128 [21] supports 64-bit hash required by Drone Remote Identification Protocol (DRIP) entity tag authentication formats and protocols for broadcast remote ID [35], which is currently being standardized, but ASCON-XOF can also be applied here. Given the usefulness of XOF, it is important to evaluate the security of ASCON-XOF.

Our Contributions. In this paper, we analyze reduced-round ASCON-XOF. We present a preimage attack on the 2-round ASCON-XOF, which improves upon Dobraunig et al.'s attack. We analyze that the attack complexity can be reduced by carefully selecting the guess bits based on a *greedy algorithm* in the context of guess-and-determine. Our attack lowers the complexity of Dobraunig et al.'s attack to 2^{34}, and we have fully implemented and verified our attack.[3] We also provide the number of guess bits required to linearize one output bit after 3- and 4-round ASCON's permutation, respectively. We then connect the results to preimage attacks on ASCON-XOF. Our contributions are summarized as follows:

1. **Analysis of the number of guess bits required to linearize one output bit after 3- and 4-round ASCON's permutation**
   Given that the algebraic degree of ASCON's S-box is 2, the algebraic degree after 3 and 4 rounds can be up to 8 and 16, respectively, so linearizing the output bits is not an easy task. Interestingly, we observe that if the linearization of the polynomials is performed carefully and the characteristics of ASCON's S-box are properly considered, the output bits can be linearized beyond 3-round to 4-round ASCON-XOF. We mainly focus on the approach of linearizing quadratic or higher-order terms with the fewest possible bit guesses. We show that the number of guess bits required to linearize one output bit after 3-/4-round ASCON-XOF is 22 and 60, respectively.
2. **First preimage attacks on 3-round ASCON-XOF with a 64-bit output**
   Preimage attacks on 3-/4-round ASCON-XOF with a 64-bit output have not been proposed thus far. Based on the number of guess bits to linearize one output bit, we perform an exhaustive search on combinations of output bits to be linearized, selecting combinations that require the least number of guess bits. In our attack on 3-round ASCON-XOF, we need to guess 56 bits to linearize 8 output bits, and then we can recover a preimage with a time complexity of 2^{56} by solving a system of linear equations. Since we need to guess 60 bits to linearize just one output bit in the 4-round ASCON's permutation, the preimage attack on ASCON-XOF is required a time complexity

[3] Recently, [23] independently developed a preimage attack on 2-round ASCON-XOF.

Table 1. Comparison of Preimage Attacks on Ascon-Xof. Our attacks on Ascon-Xof are equally applicable to Ascon-Xofa. We do not claim that 4-round Ascon-Xof with a 64- and 128-bit output is broken by our attacks, since the advantage of ours is small. MitM = Meet-in-the-Middle attack.

Rounds	Hash	Time	Memory	Method	Reference
2/12	64	2^{39}	-	Algebraic†	[10]
		$2^{31.56}$	-	Algebraic	[23]
		$\mathbf{2^{34}}$	**-**	**Algebraic‡**	**Section 3**
	128	2^{103}	-	Algebraic†	[10]
		$\mathbf{2^{98}}$	**-**	**Algebraic‡**	**Section 3**
3/12	64	$\mathbf{2^{56}}$	**-**	**Algebraic†**	**Section 4**
	128	$2^{120.58}$	2^{39}	MitM	[29]
		$2^{114.53}$	2^{30}	MitM	[30]
		$2^{112.205}$	-	Algebraic	[23]
		$\mathbf{2^{120}}$	**-**	**Algebraic†**	**Section 4**
4/12	64	$\mathbf{2^{63}}$	**-**	**Algebraic†**	**Section 4**
	128	$2^{124.67}$	2^{50}	MitM	[29]
		$2^{124.49}$	-	Algebraic	[23]
		$\mathbf{2^{127}}$	**-**	**Algebraic†**	**Section 4**

† A setting that takes into account Ascon's permutation without round constants and initialization, where IV is set to 0.
‡ A setting that takes into account Ascon's permutation without initialization, where IV is set to 0.

of 2^{63}. Since some readers may not consider this an attack, we do not claim that 4-round Ascon-Xof is broken by our attack.

Table 1 summarizes the preimage attacks on Ascon-Xof. Since our attacks can be extended to 128-bit outputs in a straightforward way, we also provide a comparison between our results and existing ones in the table. The original target of our attacks is a 64-bit output hash; for a 128-bit output hash, the complexities of our attacks are simply multiplied by 2^{64}.

Paper Organization. Section 2 describes the specifications of Ascon. Section 3 describes our preimage analysis on 2-round Ascon-Xof. Section 4 describes how to find the number of guess bits required to linearize one output bit after 3- and 4-round Ascon's permutation, respectively. We use these results to mount preimage attacks on 3- and 4-round Ascon-Xof. Section 5 presents our conclusion and discussion.

2 Description of Ascon

Ascon [11], designed by Dobraunig et al., consists of a permutation-based AEAD and hashing schemes, whose core components are the 320-bit iterative permutations p^a and p^b with a and b rounds, respectively. In hashing modes, a and b are set to 12 (for Ascon-Hasha and Ascon-Xofa, set $b = 8$), and the mode of operation is sponge construction [4] (see Figure 1). For the description and application of the round transformations, the 320-bit state is split into five 64-bit words $X_i (0 \leq i \leq 4)$ as illustrated in Figure 2, where X_0 is the MSB and X_4 the LSB.

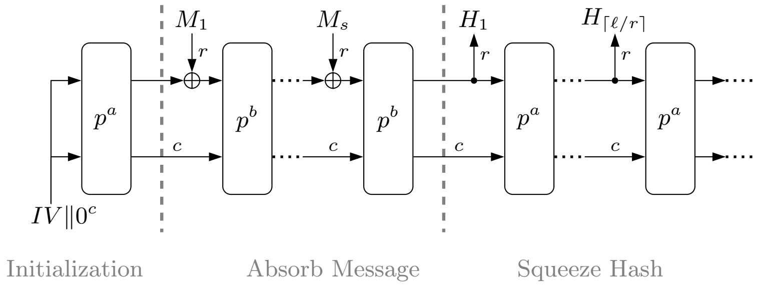

Fig. 1. Hashing modes of Ascon [13]

The 320-bit initial state of Ascon-Hash (Ascon-Hasha) and Ascon-Xof (Ascon-Xofa) is defined by a constant IV that specifies the algorithm parameters, including the rate r and round numbers a, and the value $a - b$, each written as an 8-bit integer. It is followed by the maximal output length of h bits as a 32-bit integer with $h = l = 256$ for Ascon-Hash and Ascon-Hasha, $h = 0$ for unlimited output in Ascon-Xof and Ascon-Xofa, and a 256-bit zero value. The a-round permutation p^a is applied to initialize the state S.

$$IV_{r,a,b,h} \leftarrow 0^8\|r\|a\|a-b\|h = \begin{cases} \texttt{00400C0000000100} & \text{for Ascon-Hash} \\ \texttt{00400C0400000100} & \text{for Ascon-Hasha} \\ \texttt{00400C0000000000} & \text{for Ascon-Xof} \\ \texttt{00400C0400000000} & \text{for Ascon-Xofa} \end{cases} \quad (1)$$

$$S \leftarrow p^a\left(IV_{r,a,b,h}\|0^{256}\right)$$

Ascon's hashing modes process the message M in blocks of r bits. The padding process first appends a single 1 and then the smallest number of zeroes to M such that the length of the padded message is a multiple of r bits. The padded message is split into s blocks of r bits and satisfies the following:

$$M_1, \ldots, M_S \leftarrow r\text{-bit blocks of } M\|1\|0^{r-1-(|M| \bmod r)} \quad (2)$$

The core permutation p consists of three functions: the addition of constants p_C, the substitution layer p_S, and the linear diffusion layer p_L ($p = p_L \circ p_S \circ p_C$). When representing a layer of the r-th round, we append it to the subscript; for example, the substitution layer of the 1 round is denoted by $p_{S,1}$. The input state to the round permutation at the r-th round is denoted by $X^r = X_0^r||X_1^r||X_2^r||X_3^r||X_4^r$ while the output state after the p_S layer is given by $Y^r = Y_0^r||Y_1^r||Y_2^r||Y_3^r||Y_4^r$. One bit of each word is denoted by $[\cdot]$. For example, $X_i^r[j]$ represents the j-th (starting from the left[4]) bit of word i at the r-th round for $j = 0, \cdots, 63$, and consecutive bits are denoted by $X_i^r[j, \ldots, k]$ for $0 \leq j < k \leq 63$. Throughout the paper, we implicitly represent indices as modulo 64.

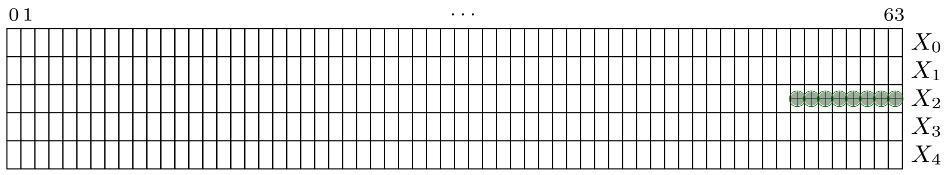

Fig. 2. Ascon's state of 320=5×64 bits.

Addition of Constants (p_C). An 8-bit constant is added to the $X_2[56, \ldots, 63]$ at each round (see Figure 2). The added constant changes depending on the round.

Substitution Layer (p_S). Each column of the 320-bit state is updated by applying the Ascon's 5-bit S-box (see Figure 3). The algebraic normal form (ANF) of the S-box is given in Equation 3. As the algebraic degree of Ascon's 5-bit S-box is 2, the algebraic degree of one round of permutation p is also 2.

$$\begin{aligned}
Y_0[j] &= X_4[j]X_1[j] \oplus X_3[j] \oplus X_2[j]X_1[j] \oplus X_2[j] \oplus X_1[j]X_0[j] \oplus X_1[j] \oplus X_0[j]\\
Y_1[j] &= X_4[j] \oplus X_3[j]X_2[j] \oplus X_3[j]X_1[j] \oplus X_3[j] \oplus X_2[j]X_1[j] \oplus X_2[j] \oplus X_1[j] \oplus X_0[j]\\
Y_2[j] &= X_4[j]X_3[j] \oplus X_4[j] \oplus X_2[j] \oplus X_1[j] \oplus 1\\
Y_3[j] &= X_4[j]X_0[j] \oplus X_4[j] \oplus X_3[j]X_0[j] \oplus X_3[j] \oplus X_2[j] \oplus X_1[j] \oplus X_0[j]\\
Y_4[j] &= X_4[j]X_1[j] \oplus X_4[j] \oplus X_3[j] \oplus X_1[j]X_0[j] \oplus X_1[j]
\end{aligned} \tag{3}$$

Linear Diffusion Layer (p_L). Each row of the 320-bit state is internally diffused by a linear transformation $\sum_i$ (see Figure 4), where $\sum_i$ is given in Equation 4. Here, $\ggg$ is the right cyclic shift operation over a 64-bit word.

[4] In the proposal paper of Ascon [11], the bit index for each word starts from the right.

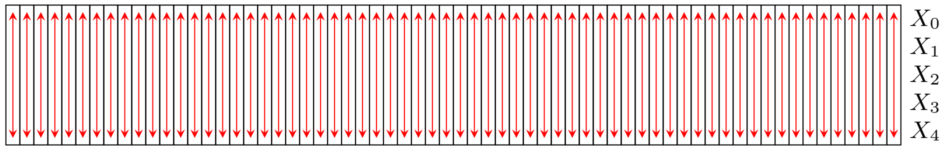

Fig. 3. Ascon's substitution layer.

$$\begin{aligned}
Y_0 &= \Sigma_0\left(Y_0\right) = Y_0 \oplus (Y_0 \ggg 19) \oplus (Y_0 \ggg 28)\\
Y_1 &= \Sigma_1\left(Y_1\right) = Y_1 \oplus (Y_1 \ggg 61) \oplus (Y_1 \ggg 39)\\
Y_2 &= \Sigma_2\left(Y_2\right) = Y_2 \oplus (Y_2 \ggg 1) \oplus (Y_2 \ggg 6)\\
Y_3 &= \Sigma_3\left(Y_3\right) = Y_3 \oplus (Y_3 \ggg 10) \oplus (Y_3 \ggg 17)\\
Y_4 &= \Sigma_4\left(Y_4\right) = Y_4 \oplus (Y_4 \ggg 7) \oplus (Y_4 \ggg 41)
\end{aligned} \tag{4}$$

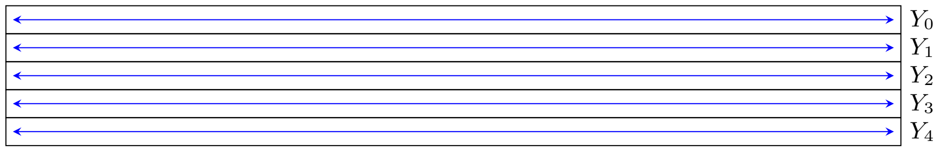

Fig. 4. Ascon's linear diffusion layer with a 64-bit diffusion function $\sum_i(Y_i)$.

Rotational Symmetry of Ascon's Permutation. We observe that the bits composing the leaked polynomial from the hash value exhibit *rotation-invariance*, in the sense that two distinct sets of bits forming the two output bits (i.e., two polynomials) become identical after a specific number of rotations. Using this property, it is sufficient to analyze only one representative bit to find the set of guess bits needed to linearize one bit after a certain round of Ascon. This property is implicitly taken into account when analyzing Ascon-Xof.

3 Preimage Attacks on 2-round Ascon-Xof

In this section, we present an attack on 2-round Ascon-Xof (with a 64-bit output) without round constants and initialization, where IV is set to 0. This setting is the same as that of the preliminary analysis of Ascon proposed by Dobraunig et al.

3.1 Dobraunig et al.'s Attack

First, we analyze the Dobraunig et al.'s attack in detail. In [10], the preimage attack on 2-round Ascon-Xof with a 64-bit output was proposed. For simplicity, they consider the round-reduced variant of Ascon-Xof without round constants and initialization, where IV is set to 0. The core of this attack is to linearize the state bits after the substitution layer in round 2 by guessing some of the input state bits in round 1. The overall configuration for their attack is presented in Figure 5, where we add $*$ to the subscript of p to indicate permutation without constants.

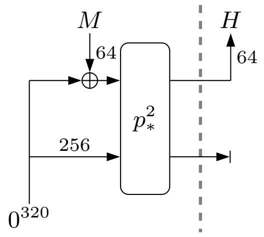

Fig. 5. Dobraunig et al.'s attack model

Let us take a closer look at the process of obtaining a preimage. As IV is set to 0, X_1^1, X_2^1, X_3^1, and X_4^1 in the input states are fixed to zero and an attacker is free to choose the bits of X_0^1. This leads to two properties regarding the input and output of Ascon's S-box.

Property 1. Let $X_1[j]$, $X_2[j]$, $X_3[j]$, and $X_4[j]$ all be zero. Then, when $X_0[j] = 0$ and $X_0[j] = 1$, $Y_0[j] = Y_1[j] = Y_3[j] = 0$ and $Y_0[j] = Y_1[j] = Y_3[j] = 1$, respectively.

Property 2. Let $X_1[j]$, $X_2[j]$, $X_3[j]$, and $X_4[j]$ all be zero. Then, $Y_2[j] = 1$ and $Y_4[j] = 0$ always hold.

Due to the word-wise structure of Ascon's linear layer, we only have to obey the first bit of the ANF of the S-box. By Property 2, we get:

$$\begin{aligned}
Y_0^2[j] &= X_4^2[j]X_1^2[j] \oplus X_3^2[j] \oplus X_2^2[j]X_1^2[j] \oplus X_2^2[j] \oplus X_1^2[j]X_0^2[j] \oplus X_1^2[j] \oplus X_0^2[j] \\
&= X_1^2[j]X_0^2[j] \oplus X_0^2[j] \oplus X_3^2[j] \oplus 1 \\
&= (Y_1^1[j] \oplus Y_1^1[j-61] \oplus Y_1^1[j-39])(Y_0^1[j] \oplus Y_0^1[j-19] \oplus Y_0^1[j-28]) \\
&\quad \oplus (Y_0^1[j] \oplus Y_0^1[j-19] \oplus Y_0^1[j-28]) \oplus (Y_3^1[j] \oplus Y_3^1[j-10] \oplus Y_3^1[j-17]) \\
&\quad \oplus 1
\end{aligned} \tag{5}$$

The only non-linear term in $Y_0^2[j]$ is $X_1^2[j]X_0^2[j]$, which can be linearized by guessing 3 bits of X_0^1. Thus, if we do not take independencies into account, we can get 1 additional linear equation in Y_0^2 per 3 guessed bits of X_0^1. After 48 guesses, we get 16 linear equations and can attempt to solve the system and get a candidate solution for H. This can be further improved by considering dependencies. For instance, if 39 consecutive bits in X_0^1 are guessed, already 25

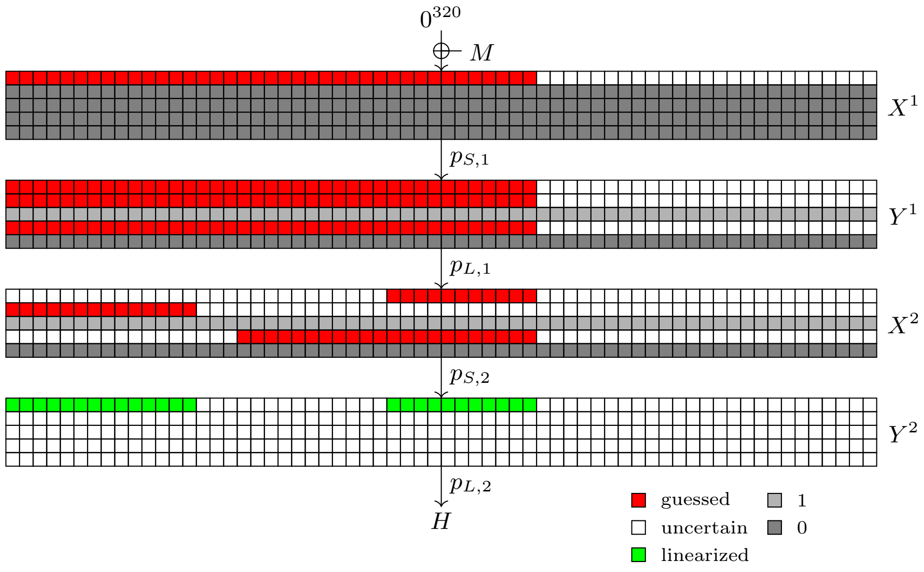

Fig. 6. Dobraunig et al.'s attack

linear equations can be derived (see Figure 6 for an example of simply guessing 39 consecutive bits of $X_0^1[0,\ldots,38]$). Taking the padding bit into account, we have 63 degrees of freedom. Furthermore, the remaining 25 bits of X_0^1 that are not guessed are included as variables in the set of 25 linear equations. Consequently, on average, one block solution for a preimage can be found with 2^{39} guesses.

Remark 1. Since the matrix representing the linear diffusion layer of Ascon has full rank, Y_0^2 is uniquely determined according to H.

3.2 Our Improved Attack

Here, we propose an improved preimage attack on 2-round Ascon-Xof. Taking the padding bit into account, we have only 63 degrees of freedom in our attacks, as in [10]. It should be noted that this consideration applies equally to Section 4.

We first elaborate on the selection of the guess bits used in the attack. In fact, the best attack strategy is to guess fewer bits and obtain as many bits as possible to be linearized. Instead of the previous method of guessing consecutive bits, we try to construct a more efficient attack using the greedy algorithm-based guessing bit selection method.

Greedy Algorithm. A greedy algorithm is an algorithm that follows a problem solving heuristic for local optimal selection at each step to find a global optimal value. For many problems, this algorithm does not always produce an optimal solution, but nevertheless it can produce a solution that approximates a globally optimal solution in a reasonable amount of time. This algorithm has been applied to select the H-representation of the convex hull of all possible differential patterns of the S-box [32], and many cryptographic applications have been developed since then. We use the greedy algorithm to find a set of guess bits suitable for the preimage attack on 2-round Ascon-Xof.

Observation for Improvement. Dobraunig et al. simply select the guess bits consecutively and then compute the number of linearized polynomials. It should be noted that in Equation 5, for $Y_0^2[j]$ to be linearized, it is sufficient for only one of $X_0^2[j]$ and $X_1^2[j]$ to be a constant. According to the word-wise structure of Ascon's linear layer, the guess bits required to make $X_0^2[j]$ and $X_1^2[j]$ constant are widely spread over Y_0^1 and Y_1^1. If the guess bits of X_0^1 are dense, there may be a case where only some of the 3 bits of Y_0^1 (resp. Y_1^1) required to make $X_0^2[j]$ (resp. $X_1^2[j]$) a constant are determined and the remaining bits must be guessed. Thus, if the guess bits of X_0^1 are selected consecutively, the bits of Y_0^2 are not linearized as much as expected.

We first recall that the attacker can select the 63 bits corresponding to the message M_1 for the attack on Ascon-Xof (i.e., X_0^1 can be adjusted, except for the padding bit). When selecting 39 guess bits, the entire space of $\binom{63}{39} \approx 2^{57.1}$ must be searched. Furthermore, if the guess bits are reduced to less than 39 bits, the number of search cases increases because the maximum value of $\binom{n}{r}$ occurs at $(n-1)/2$ or at $(n+1)/2$ when n is odd. We overcome this by designing an algorithm based on the greedy algorithm for selecting guess bits. The greedy algorithm helps us to linearize as many bits as we want in Y_0^2, which is done by finding a set with an index of the minimum bits required based on each 64-bit position of X_0^1. Here, we have to look carefully at the first two rows of $p_{L,1}$. If we want $X_0^2[j]$ to be a constant, we need to guess the bits of $X_0^1[j]$, $X_0^1[j-19]$, and $X_0^1[j-28]$, and if we want $X_1^2[j]$ to be a constant, we have to guess the bits of $X_0^1[j]$, $X_0^1[j-61]$, and $X_0^1[j-39]$. A case where $X_0^2[j]$ and $X_1^2[j]$ are both constants can be considered, but it is not as effective in the overall attack because it is necessary to guess five bits of X_0^1 for one bit linearization. For an effective attack, we consider the case where only one of them is a constant. Consequently, we selected either $X_0^2[j]$ or $X_1^2[j]$ to be a constant that minimizes the number of guess bits in X_0^1 needed to linearize $Y_0^2[j]$ at each step of the greedy algorithm. Algorithm 1 presents the process of selecting guess bits to attack 2-round Ascon-Xof using the greedy algorithm in detail.

The following is a closer look at Algorithm 1.

1. In Step 5, we include 1 bit of Y_0^2 in the candidate set $\mathcal{L}_C$ of the guess bits first, and in step 7, we include 3-bit indices of X_0^1 necessary to linearize the (included) bit of $\mathcal{L}_C$ in $\mathcal{B}_C$.
2. In Steps 8–14, we select the bit of Y_0^2 that becomes linear by guessing the minimum bit of X_0^1 at every moment through the greedy algorithm, and we include the indices of 3 bits of X_0^1 needed to make it constant in $\mathcal{B}_C$. We repeat this process and stop selecting bits when $|\mathcal{L}_C| = \mathcal{N}$ holds.

Algorithm 1: Process of selecting guess bits to attack 2-round Ascon-Xof using the greedy algorithm

```
Input: $\mathcal{N}$: Number of bits of $Y_0^2$ to be linearized ($1 < \mathcal{N} < 64$)
Output: $\mathcal{B}$: Set of bit indices to be guessed in $X_0^1$
    $\mathcal{L}$: Set of bit indices to be linearized in $Y_0^2$
 1: $\mathcal{I} := \{0, 1, \ldots, 63\}$
 2: $\mathcal{B} := \emptyset,\ \mathcal{B}_{all} := \emptyset,\ \mathcal{L} := \emptyset,\ \mathcal{L}_{all} := \emptyset$
 3: for $j \in \mathcal{I}$ do
 4:     $\mathcal{T} := \mathcal{I},\ \mathcal{B}_C := \emptyset,\ \mathcal{L}_C := \emptyset$
 5:     $\mathcal{L}_\mathcal{C} \leftarrow \mathcal{L}_\mathcal{C} \cup \{j\}$
 6:     $\mathcal{T} \leftarrow \mathcal{T} - \{j\}$
 7:     $\mathcal{B}_C \leftarrow \mathcal{B}_C \cup \{j\} \cup \{j-19\} \cup \{j-28\}$
 8:     for $cnt \in \{0, 1, \ldots, \mathcal{N}-2\}$ do
 9:         $\mathcal{M} \leftarrow$ The bit index of $Y_0^2$ that can be linearized by guessing the least bit of
            $X_0^1$ ($\mathcal{M} \in \mathcal{T}$)
10:         $\mathcal{M}_{bit} \leftarrow$ The bit indices of $X_0^1$ that linearize $\mathcal{M}$ of $Y_0^2$
11:         $\mathcal{B}_C \leftarrow \mathcal{B}_C \cup \{\mathcal{M}_{bit}\}$
12:         $\mathcal{L}_\mathcal{C} \leftarrow \mathcal{L}_\mathcal{C} \cup \{\mathcal{M}\}$
13:         $\mathcal{T} \leftarrow \mathcal{T} - \{\mathcal{M}\}$
14:     end for
15:     for $k \in \mathcal{I} - \mathcal{L}_C$ do
16:         if $Y_0^2[k]$ is linearized because of the elements of $\mathcal{B}_C$ then
17:             $\mathcal{L}_\mathcal{C} \leftarrow \mathcal{L}_\mathcal{C} \cup \{k\}$
18:         end if
19:     end for
20:     if there is any one of the bit indices of $\mathcal{I} - \mathcal{B}_C$ that cannot be included as a
        variable in the linear equations generated from the bits of $\mathcal{L}_C$ then
21:         continue
22:     else
23:         $\mathcal{B}_{all} \leftarrow \mathcal{B}_{all} \cup \{\mathcal{B}_C\}$
24:         $\mathcal{L}_{all} \leftarrow \mathcal{L}_{all} \cup \{\mathcal{L}_C\}$
25:     end if
26: end for
27: Min $\leftarrow$ Minimum size of elements of $\mathcal{B}_{all}$
28: Max $\leftarrow$ Of the elements of $\mathcal{B}_{all}$ whose size is Min, the maximum size of the
    corresponding elements of $\mathcal{L}_{all}$
29: $\mathcal{B} \leftarrow$ Element of $\mathcal{B}_{all}$ that has a size Min and has a size Max of an element of $\mathcal{L}_{all}$
    in that index
30: $\mathcal{L} \leftarrow$ Element of $\mathcal{L}_{all}$ that correspond to $\mathcal{B}$
31: return $\mathcal{B}, \mathcal{L}$
```

3. In Steps 15–19, we check whether any bits of Y_0^2 excluding the selected bits of $\mathcal{L}_C$ can be linearized through $\mathcal{B}_C$, and if so, include the index of the bits in $\mathcal{L}_C$.
4. In Steps 20–25, we check whether the two sets $\mathcal{B}$ and $\mathcal{L}$ can be used for attacks. First, in Steps 20–21, we check whether the bits of $\mathcal{I} - \mathcal{B}_C$ that were not guessed in X_0^1 are included as variables in the linear equations

generated in $\mathcal{L}$. If included, include $\mathcal{B}_C$ and $\mathcal{L}_C$ as elements of $\mathcal{B}_{all}$ and $\mathcal{L}_{all}$, respectively, and proceed to the next step. If not included, we repeat from Step 8 again.

5. In Step 27, we set the smallest size of the element $\mathcal{B}_{all}$, the set of bit indices to be guessed from X_0^1, as Min. In Step 28, we set the maximum size of the set of the elements of $\mathcal{L}_{all}$ with respect to the elements of size Min of $\mathcal{B}_{all}$ as Max.
6. In Steps 29–30, while having size Min in $\mathcal{B}_{all}$, the element of Max size of $\mathcal{L}_{all}$ of the corresponding index is set to $\mathcal{B}$. The corresponding element of $\mathcal{L}_{all}$ is set to $\mathcal{L}$.

We performed Algorithm 1 with $\mathcal{N}$ set to 31, and when $j = 25$, the best solution was provided by guessing 34 bits. For the remaining j, most of the bits in the X_0^1 position of $\mathcal{I} - \mathcal{B}_C$ were not included as variables of the linear equations, or more than 35 bit guesses were required to find a preimage. $\mathcal{B}$ and $\mathcal{L}$ (both $j = 25$) were found through this process, as presented in Table 2.

Table 2. Bits information for the preimage attack on 2-round Ascon-Xof

Guess bits of X_0^1 ($\mathcal{B}$, 34 bits)	0, 1, 3, 4, 6, 7, 9, 10, 12, 15, 16, 18, 19, 21, 24, 25, 27, 28, 29, 30, 31, 33, 34, 35, 37, 38, 40, 43, 44, 46, 49, 52, 55, 61
Linearized bits of Y_0^2 ($\mathcal{L}$, 31 bits)	0, 1, 3, 4, 6, 7, 9, 10, 12, 15, 16, 18, 19, 21, 24, 25, 27, 28, 29, 30, 31, 34, 35, 37, 38, 40, 43, 44, 46, 49, 52

For sets $\mathcal{B}$ and $\mathcal{L}$ consisting of found bit indices, the process of finding a preimage for H_1 is as follows.

1. **Preprocessing phase:** Store sets of bit indices of X_0^1 that must be guessed for $X_0^2[j]$, $X_1^2[j]$, and $X_3^2[j]$ to become constants as elements of T_{X_0}, T_{X_1}, and T_{X_3}, where each is a set having 64 sets of 3 elements as elements. For example, $T_{X_0}(3)$ is a set of bit indices of X_0^1 that must be guessed in order for $X_0^2[3]$ to be a constant.
2. When constructing linear equations in a later process, information on bit indices of X_0^1 (especially, non-guessed bit indices) is required to calculate $X_0^2[j]$, $X_1^2[j]$, and $X_3^2[j] (j \in \mathcal{L})$. Therefore, store these information in U_{X_0}, U_{X_1}, and U_{X_3}, respectively. For example, $U_{X_0}(3)$ is a set of bit indices of X_0^1 that must be guessed to make $X_0^2[3]$ constant, but are not guessed. Note that at least one of $U_{X_0}(j)$ and $U_{X_1}(j)$ is empty, because $X_0^2[j]$ or $X_1^2[j]$ must be constant.
3. Y_0^2 is calculated from the hash value H through $p_{L,2}^{-1}$.
4. **Guess phase:** A possible value $j \in \{0, \ldots 2^{34} - 1\}$ is fixed at the bit indices in set $\mathcal{B}$ ($|\mathcal{B}| = 34$). Based on this and Y_0^2, linear equations are generated according to Equation 5 at the indices of $\mathcal{L}$. As $|\mathcal{L}| = 31$ and $|\mathcal{B}| = 34$ hold, 31 linear equations with 30 variables are created.

5. We check whether the constructed linear equation system is unsolvable. If it is unsolvable, we return to Step 4 and perform the same process for other j values. If the system of linear equations is solvable, we go to Step 6.
6. Gauss–Jordan elimination is performed on the constructed system of linear equations. If there is a free variable, 30 bits are determined for every number of cases; if not, uniquely determined.
7. 2-round Ascon-Xof is run with a message that combines 30 determined bits and 34 guessed bits, and we check if the calculated hash value matches H. If the hash value does not match, we go back to Step 4 and perform the same process for another j value.

It is crucial to verify that the linear equations constructed include all the variables corresponding to the unknown bits of X_0^1, except for those guessed. This is because, if these bits are not included as variables of the linear equations, they cannot be determined even if the linear equation system is solved by Gauss–Jordan elimination, and the corresponding bits must be calculated as the complexity that the attacker must guess. Our attack is valid because 30 bits, excluding the guessed bits of X_0^1, are all included as variables of 31 linear equations. If we perform 2^{34} guesses on the 34 bits of X_0^1, we can expect to find a one-block solution for the preimage (see Figure 7). Our attack has been fully implemented and verified, and we have made the attack tool publicly available at https://github.com/pion2er/Preimage_Attacks_on_Ascon_2r.

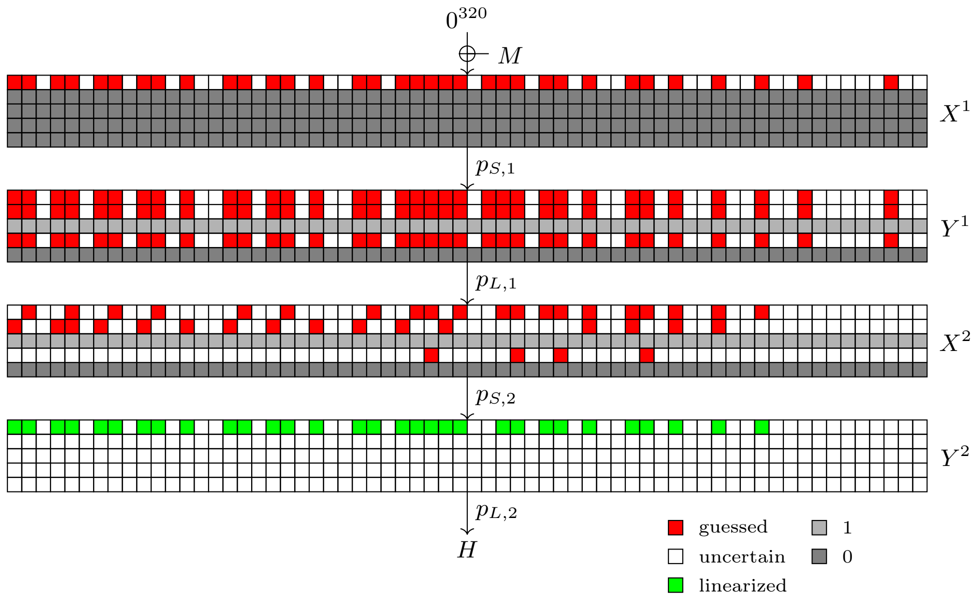

Fig. 7. Our improved preimage attack on 2-round Ascon-Xof

3.3 Mitigating Assumption.

Thus far, we have analyzed the attack process for 2-round Ascon-Xof without round constants and initialization, where IV is set to 0. However, according to our analysis, considering the round constants does not affect the overall complexity of the preimage attack. When we consider $p_{C,1}$ and $p_{C,2}$, which are addition of constants, for Ascon-Hash and Ascon-Xof (resp. Ascon-Hasha and Ascon-Xofa), the 1-byte constant added at $p_{C,1}$ is 0xf0 (resp. 0xb4), and for $p_{C,2}$, is 0xe1 (resp. 0xa5). The hamming weight of all 1-byte round constants of Ascon is 4, so we have to deal with these 4 bits appropriately. For Ascon-Hash and Ascon-Xof, if bits of $X_0^1[56, \ldots, 59]$ (resp. $Y_0^2[56, \ldots, 58]$, $Y_0^2[63]$), which are Hamming weight indices corresponding to constant 0xf0 (resp. 0xe1), are not included in $\mathcal{B}$ (resp. $\mathcal{L}$), the preimage for hash can be found in the same way as before. One thing to note is that when $\mathcal{B}$ and $\mathcal{L}$ in Table 2 are used, $Y_0^2[0]$ cannot be made linear. This is because $X_2^2[0]$ is not determined to be 1, because the round constant is XORed on $X_0^1[58]$, leaving $Y_2^1[58]$ as an unknown number, not 1. However, given that 30 linear equations are still generated by 34 bits guessing, and 30 bits that are not guessed remain as variables of the linear equations, the validity of the preimage attack is not affected. For Ascon-Hasha and Ascon-Xofa, there are no sets of bit indices ($\mathcal{B}$ and $\mathcal{L}$) satisfying the above method by guessing 34 bits (the size of $\mathcal{B}$ must be 36 to satisfy this). However, the preimage attack can be performed with the same complexity by including some bits of X_0^1 corresponding to the Hamming weights of constants as guess bits.

4 Revealing the Level of Linearization of Ascon's Permutation for 3 and 4 Rounds

In this section, we reveal the level of linearization of Ascon's permutation for 3 and 4 rounds. We then try to mount these results to preimage attacks on 3- and 4-round Ascon-Xof. The overall framework of this attack is the same as described in Section 3, in which the guess bits of X_0^1 are selected and the non-guessed bits are determined by solving the linear equation system that can be generated in Y_0^2.

4.1 Preimage Attack on 3-round Ascon-Xof

We first have to choose the guess bits of X_0^1 so that we can make the bits of Y_0^3 linear. The difficulty is that the algebraic degree of Ascon's S-box is 2, so the algebraic degree after 3 rounds can be up to 8. To linearize a non-linear term with the algebraic degree of 8, we need to guess many bits of X_0^1. Nevertheless, this obstacle can be overcome by exploiting Ascon's S-box properties and effectively linearizing the non-linear term of the polynomial that is leaked by the hash value. Now, we show how many bits of X_0^1 must be guessed to linearize one bit of Y_0^3.

Equation 6 can be obtained by expressing $Y_0^3[j]$ $(0 \leq j \leq 63)$ in the ANF of Ascon's S-box.

$$\begin{aligned} Y_0^3[j] &= X_4^3[j]X_1^3[j] \oplus X_3^3[j] \oplus X_2^3[j]X_1^3[j] \oplus X_2^3[j] \oplus X_1^3[j]X_0^3[j] \oplus X_1^3[j] \oplus X_0^3[j] \\ &= (X_4^3[j] \oplus X_2^3[j] \oplus X_0^3[j] \oplus 1)X_1^3[j] \oplus X_3^3[j] \oplus X_2^3[j] \oplus X_0^3[j] \end{aligned} \tag{6}$$

We focus on making $X_1^3[j]$ constant, by applying $p_{L,2}^{-1}$ to X_1^3 and expressing it as the ANF of Ascon's S-box, resulting in Equation 7.

$$\begin{aligned} X_1^3[j] &= Y_1^2[j] \oplus Y_1^2[j-61] \oplus Y_1^2[j-39] \\ &= (\underbrace{X_3^2[j]X_1^2[j] \oplus X_0^2[j]}_{T_{0,j}} \oplus 1) \oplus (\underbrace{X_3^2[j-61]X_1^2[j-61] \oplus X_0^2[j-61]}_{T_{1,j}} \oplus 1) \\ &\quad \oplus (\underbrace{X_3^2[j-39]X_1^2[j-39] \oplus X_0^2[j-39]}_{T_{2,j}} \oplus 1) \end{aligned} \tag{7}$$

Let $S_{T_{0,j}}$, $S_{T_{1,j}}$, and $S_{T_{2,j}}$ be the sets containing the bits of X_0^1 that must be guessed to make $T_{0,j}$, $T_{1,j}$, and $T_{2,j}$ constants. If we guess the bits of X_0^1 in the position of the union of $S_{T_{0,j}}$, $S_{T_{1,j}}$, and $S_{T_{2,j}}$, we can make $T_{0,j}$, $T_{1,j}$, and $T_{2,j}$ constants, and $X_1^3[j]$ is also determined to be constant. Note that as Ascon's permutation is rotation-invariant, we consider only the case of $j = 0$ as a representative case. The bit indices to be guessed when $j = 0$ are presented in Table 3.

Table 3. Indices of guess bits of X_0^1 $(j = 0)$

$S_{T_{0,0}}(7)$	0, 3, 25, 36, 45, 47, 54
$S_{T_{1,0}}(7)$	3, 6, 28, 39, 48, 50, 57
$S_{T_{2,0}}(7)$	6, 8, 15, 25, 28, 50, 61
$S_{T_{0,0}} \cup S_{T_{1,0}} \cup S_{T_{2,0}}(16)$	0, 3, 6, 8, 15, 25, 28, 36, 39, 45, 47, 48, 50, 54, 57, 61

Let us guess the 16 bits of $S_{T_{0,j}} \cup S_{T_{1,j}} \cup S_{T_{2,j}}$ and denote the constant $X_1^3[j]$ as C. Then, by Property 2, Equation 6 is rewritten as follows.

$$
\begin{aligned}
Y_0^3[j] &= (X_4^3[j] \oplus X_2^3[j] \oplus X_0^3[j] \oplus 1)C \oplus X_3^3[j] \oplus X_2^3[j] \oplus X_0^3[j] \\
&= \{(Y_4^2[j] \oplus Y_4^2[j-7] \oplus Y_4^2[j-41]) \oplus (Y_2^2[j] \oplus Y_2^2[j-1] \oplus Y_2^2[j-6]) \\
&\quad \oplus (Y_0^2[j] \oplus Y_0^2[j-19] \oplus Y_0^2[j-28]) \oplus 1\}C \oplus (X_3^2[j] \oplus X_3^2[j-10] \oplus X_3^2[j-17]) \\
&\quad \oplus (X_0^2[j] \oplus X_0^2[j-1] \oplus X_0^2[j-6]) \oplus (X_0^2[j] \oplus X_0^2[j-19] \oplus X_0^2[j-28]) \\
&= [\{(X_1^2[j]X_0^2[j] \oplus X_1^2[j] \oplus X_3^2[j]) \oplus (X_1^2[j-7]X_0^2[j-7] \oplus X_1^2[j-7] \oplus X_3^2[j-7]) \\
&\quad \oplus (X_1^2[j-41]X_0^2[j-41] \oplus X_1^2[j-41] \oplus X_3^2[j-41])\} \oplus (X_1^2[j] \oplus X_1^2[j-1] \oplus X_1^2[j-6]) \\
&\quad \oplus \{(X_1^2[j]X_0^2[j] \oplus X_0^2[j] \oplus X_3^2[j] \oplus 1) \oplus (X_1^2[j-19]X_0^2[j-19] \oplus X_0^2[j-19] \oplus X_3^2[j-19] \oplus 1) \\
&\quad \oplus (X_1^2[j-28]X_0^2[j-28] \oplus X_0^2[j-28] \oplus X_3^2[j-28] \oplus 1)\} \oplus 1]C \\
&\quad \oplus \{(X_3^2[j]X_0^2[j] \oplus X_0^2[j] \oplus X_1^2[j] \oplus X_3^2[j] \oplus 1) \\
&\quad \oplus (X_3^2[j-10]X_0^2[j-10] \oplus X_0^2[j-10] \oplus X_1^2[j-10] \oplus X_3^2[j-10] \oplus 1) \\
&\quad \oplus (X_3^2[j-17]X_0^2[j-17] \oplus X_0^2[j-17] \oplus X_1^2[j-17] \oplus X_3^2[j-17] \oplus 1)\} \\
&\quad \oplus (X_1^2[j] \oplus X_1^2[j-1] \oplus X_1^2[j-6]) \oplus \{(X_1^2[j]X_0^2[j] \oplus X_0^2[j] \oplus X_3^2[j] + 1) \\
&\quad \oplus (X_1^2[j-19]X_0^2[j-19] \oplus X_0^2[j-19] \oplus X_3^2[j-19] \oplus 1) \\
&\quad \oplus (X_1^2[j-28]X_0^2[j-28] \oplus X_0^2[j-28] \oplus X_3^2[j-28] \oplus 1)\} \\
&= [\underbrace{(X_1^2[j-7]X_0^2[j-7]}_{T_{3,j}} \oplus X_1^2[j-7] \oplus X_3^2[j-7]) \\
&\quad \oplus \underbrace{(X_1^2[j-41]X_0^2[j-41]}_{T_{4,j}} \oplus X_1^2[j-41] \oplus X_3^2[j-41]) \\
&\quad \oplus X_1^2[j-1] \oplus X_1^2[j-6] \oplus C \oplus \{\underbrace{(X_1^2[j-19]X_0^2[j-19]}_{T_{5,j}} \oplus X_0^2[j-19] \oplus X_3^2[j-19]) \\
&\quad \oplus \underbrace{(X_1^2[j-28]X_0^2[j-28]}_{T_{6,j}} \oplus X_0^2[j-28] \oplus X_3^2[j-28])\}]C \\
&\quad \oplus \{C \oplus \underbrace{(X_3^2[j-10]X_0^2[j-10]}_{T_{7,j}} \oplus X_0^2[j-10] \oplus X_1^2[j-10] \oplus X_3^2[j-10]) \\
&\quad \oplus \underbrace{(X_3^2[j-17]X_0^2[j-17]}_{T_{8,j}} \oplus X_0^2[j-17] \oplus X_1^2[j-17] \oplus X_3^2[j-17])\} \\
&\quad \oplus C \oplus \{\underbrace{(X_1^2[j-19]X_0^2[j-19]}_{T_{9,j}} \oplus X_0^2[j-19] \oplus X_3^2[j-19]) \\
&\quad \oplus \underbrace{(X_1^2[j-28]X_0^2[j-28]}_{T_{10,j}} \oplus X_0^2[j-28] \oplus X_3^2[j-28])\}
\end{aligned} \tag{8}
$$

Note that there are eight parts denoted by $T_{i,j}$ $(3 \le i \le 10)$ in the above expression. As $T_{i,j}$ consists of quadratic terms, it is possible to linearize $Y_0^3[j]$ if we make those terms linear. Given that 16 bits of the set $S_{T_{0,0}} \cup S_{T_{1,0}} \cup S_{T_{2,0}}$ are guessed, the indices of the guess bits of X_0^1 required to make each $T_{i,0}$ constant can be calculated, as presented in Table 4. This result depends on the nature of Ascon's linear diffusion layer and substitution layer, and 4 of the 8 quadratic terms can be made linear even by guessing the bits of the set $S_{T_{0,0}} \cup S_{T_{1,0}} \cup S_{T_{2,0}}$.

Table 4. Indices of additional guess bits of X_0^1 ($j = 0$)

$S_{T_{3,0}}(2)$	18, 60
$S_{T_{4,0}}(2)$	23, 26
$S_{T_{5,0}}(0)$	-
$S_{T_{6,0}}(0)$	-
$S_{T_{7,0}}(2)$	26, 35
$S_{T_{8,0}}(1)$	19
$S_{T_{9,0}}(0)$	-
$S_{T_{10,0}}(0)$	-
$S_{T_{3,0}} \cup S_{T_{4,0}} \cup S_{T_{7,0}} \cup S_{T_{8,0}}(6)$	18,19,23,26,35,60

As a result, by further guessing the six bits of $S_{T_{3,0}} \cup S_{T_{4,0}} \cup S_{T_{7,0}} \cup S_{T_{8,0}}$, eight quadratic terms $T_{i,0}$ ($3 \leq i \leq 10$) can be made constant, so that all terms constituting $Y_0^3[j]$ become linear terms. Therefore, if 22 bits of X_0^1 are guessed, $Y_0^3[j]$ becomes linear, which is shown in Property 3.

Property 3. For Ascon's permutation, when $X_1^1, X_2^1, X_3^1, X_4^1$ and the round constants are set to 0, 22 bits of X_0^1 must be guessed to linearize one bit of Y_0^3.

Based on this property, we first linearize the 4-th bit of the 64 bits of Y_0^3 by guessing the corresponding 22 bits of X_0^1. We then examine the number of guessed bits of X_0^1 according to all combinations of bits to be linearized among the remaining bits of Y_0^3. Specifically, we tried increasing the number of linearization bits of Y_0^3 from 3 to 10 to find the number of guess bits of X_0^1 required. We found that linearizing 8 bits required 56 bits of guessing as presented in Table 5, which gives us the lowest attack complexity. If we perform 2^{56} guesses on the 56 bits of X_0^1, we can expect to find a one-block solution for the preimage (see Figure 8). Our attack is valid because the not-guessed bits of X_0^1 are included as variables of the 8 linear equations.

4.2 Preimage Attack on 4-round Ascon-Xof

Here, we reveal to what extent the 4-round Ascon's permutation is linearized. As the algebraic degree is possible up to 16 after 4 rounds of the permutation, the linearization process is more complicated than the analysis in Section 4.1.

After 4 rounds, one bit Y_0^4 of the output is expressed as:

$$\begin{aligned} Y_0^4[j] &= X_4^4[j]X_1^4[j] \oplus X_3^4[j] \oplus X_2^4[j]X_1^4[j] \oplus X_2^4[j] \oplus X_1^4[j]X_0^4[j] \oplus X_1^4[j] \oplus X_0^4[j] \\ &= (X_4^4[j] \oplus X_2^4[j] \oplus X_0^4[j] \oplus 1)X_1^4[j] \oplus X_3^4[j] \oplus X_2^4[j] \oplus X_0^4[j] \end{aligned} \tag{9}$$

Table 5. Bits information for the preimage attack on 3-round Ascon-Xof

Guess bits of X_0^1 (56 bits)	0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 35, 36, 38, 39, 40, 41, 42, 43, 44, 45, 46, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 61, 62
Linearized bits of Y_0^3 (8 bits)	4, 5, 7, 17, 30, 33, 36, 49

In Equation 9, we need to first make $X_1^4[j]$ a constant to reduce the degree of the overall equation. $X_1^4[j]$ is expressed by the linear layer operation as follows:

$$X_1^4[j] = Y_1^3[j] \oplus Y_1^3[j-61] \oplus Y_1^3[j-39] \tag{10}$$

As Ascon's permutation is rotation-invariant, determining the guess bits of X_0^1 to make $Y_1^3[j]$ a constant will also reveal the corresponding bits of $Y_1^3[j-61]$ and $Y_1^3[j-39]$. We also focus on the case of $j=0$, which is representative of the other cases we consider, consequently making $Y_0^4[0]$ a constant (accordingly, we need to make $Y_1^3[0]$, $Y_1^3[3]$, and $Y_1^3[25]$ constant). Let us consider $Y_1^3[0]$.

$$\begin{aligned} Y_1^3[0] &= X_4^3[0] \oplus X_3^3[0]X_2^3[0] \oplus X_3^3[0]X_1^3[0] \oplus X_3^3[0] \oplus X_2^3[0]X_1^3[0] \oplus X_2^3[0] \oplus X_1^3[0] \oplus X_0^3[0] \\ &= (X_2^3[0] \oplus X_1^3[0])X_3^3[0] \oplus (X_1^3[0] \oplus 1)X_2^3[0] \oplus X_4^3[0] \oplus X_3^3[0] \oplus X_1^3[0] \oplus X_0^3[0] \end{aligned} \tag{11}$$

To reduce the degree of the overall equation, we need to make both $X_3^3[0]$ and $X_2^3[0]$ constant. By Property 2, Equations 12 and 13 hold.

$$\begin{aligned} X_3^3[0] &= Y_3^2[0] \oplus Y_3^2[54] \oplus Y_3^2[47] \\ &= (\underbrace{X_3^2[0]X_0^2[0] \oplus X_3^2[0] \oplus X_1^2[0] \oplus X_0^2[0]}_{T_0} \oplus 1) \\ &\oplus (\underbrace{X_3^2[54]X_0^2[54] \oplus X_3^2[54] \oplus X_1^2[54] \oplus X_0^2[54]}_{T_1} \oplus 1) \\ &\oplus (\underbrace{X_3^2[47]X_0^2[47] \oplus X_3^2[47] \oplus X_1^2[47] \oplus X_0^2[47]}_{T_2} \oplus 1) \end{aligned} \tag{12}$$

$$\begin{aligned} X_2^3[0] &= Y_2^2[0] \oplus Y_2^2[63] \oplus Y_2^2[58] \\ &= \underbrace{X_1^2[0] \oplus X_1^2[63] \oplus X_1^2[58]}_{T_3} \end{aligned} \tag{13}$$

Let S_{T_0}, S_{T_1}, S_{T_2} and S_{T_3} be the sets containing the bits of X_0^1 that must be guessed to make T_0, T_1, T_2 and T_3 constants. If we guess the bits of X_0^1 in the position of the union of S_{T_0}, S_{T_1}, S_{T_2} and S_{T_3}, we can make both $X_3^3[0]$ and $X_2^3[0]$ constant. The bit indices to be guessed are presented in Table 6.

Refer to the description of Property 3 for calculating the guess bits required to make $X_1^3[0]$ a constant. $X_1^3[0]$ can be made constant by additionally guessing

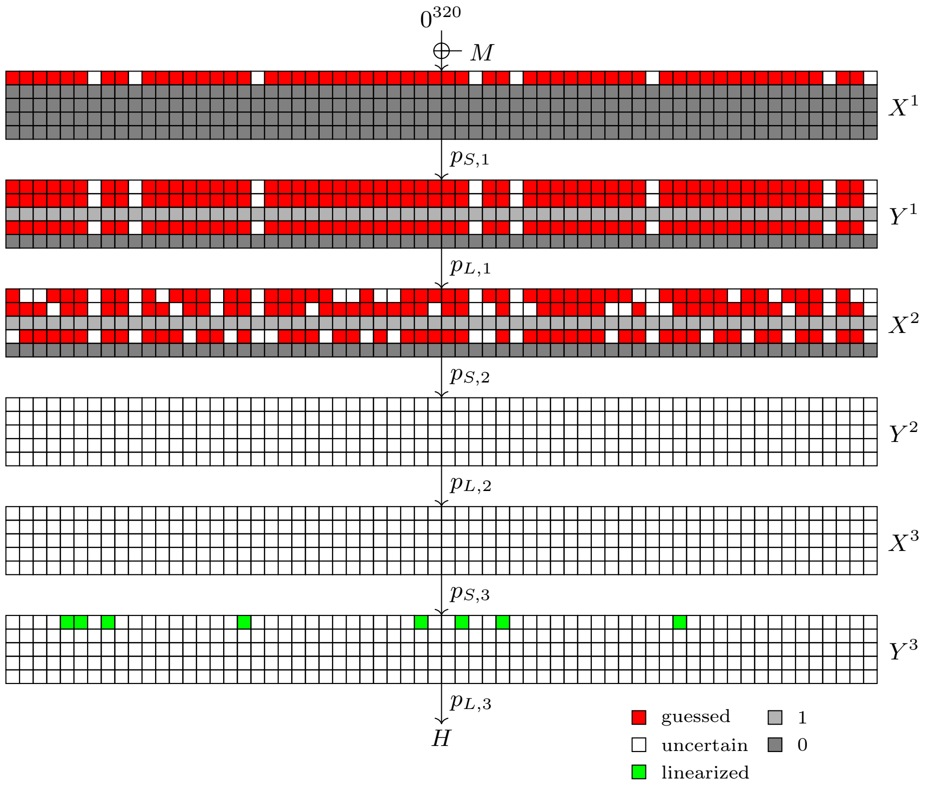

Fig. 8. Our preimage attack on 3-round Ascon-Xof

the three bits of the 6, 39 and 48 indices in addition to the 23 bits already guessed. Thus, if $X_3^3[0]$, $X_2^3[0]$, and $X_1^3[0]$ are set to C_1, C_2, and C_3, respectively, the following equation is obtained by guessing 26 bits of X_0^1.

$$\begin{aligned} Y_1^3[0] &= (X_2^3[0] \oplus X_1^3[0])X_3^3[0] \oplus (X_1^3[0] \oplus 1)X_2^3[0] \oplus X_4^3[0] \oplus X_3^3[0] \oplus X_1^3[0] \oplus X_0^3[0] \\ &= X_4^3[0] \oplus X_0^3[0] \oplus C_1C_2 \oplus C_1C_3 \oplus C_2C_3 \oplus C_1 \oplus C_2 \oplus C_3 \end{aligned} \tag{14}$$

By Property 2, Equation 15 holds.

Table 6. Indices of guess bits of X_0^1

$S_{T_0}(7)$	0, 3, 25, 36, 45, 47, 54
$S_{T_1}(7)$	15, 26, 35, 37, 44, 54, 57
$S_{T_2}(7)$	8, 19, 28, 30, 37, 47, 50
$S_{T_3}(7)$	0, 2, 3, 19, 24, 25, 58, 61, 63
$S_{T_0} \cup S_{T_1} \cup S_{T_2} \cup S_{T_3}(23)$	0, 2, 3, 8, 15, 19, 24, 25, 26, 28, 30, 35, 36, 37, 44, 45, 47, 50, 54, 57, 58, 61, 63

$$\begin{aligned}
X_4^3[0] \oplus X_0^3[0] &= Y_4^2[0] \oplus Y_4^2[57] \oplus Y_4^2[23] \oplus Y_0^2[0] \oplus Y_0^2[45] \oplus Y_0^2[36] \\
&= (\underbrace{X_1^2[0]X_0^2[0] \oplus X_3^2[0] \oplus X_0^2[0]}_{T_4} \oplus 1) \\
&\oplus (\underbrace{X_1^2[57]X_0^2[57] \oplus X_3^2[57] \oplus X_0^2[57]}_{T_5} \oplus 1) \\
&\oplus (\underbrace{X_1^2[23]X_0^2[23] \oplus X_3^2[23] \oplus X_0^2[23]}_{T_6} \oplus 1) \\
&= (\underbrace{X_1^2[0]X_0^2[0] \oplus X_1^2[0] \oplus X_3^2[0]}_{T_7} \oplus 1) \\
&\oplus (\underbrace{X_1^2[45]X_0^2[45] \oplus X_1^2[45] \oplus X_3^2[45]}_{T_8} \oplus 1) \\
&\oplus (\underbrace{X_1^2[36]X_0^2[36] \oplus X_1^2[36] \oplus X_3^2[36]}_{T_9} \oplus 1)
\end{aligned} \tag{15}$$

Note that there are six parts denoted by T_i $(4 \leq i \leq 9)$ in the above expression. We need to make these six terms constant to make $X_4^3[0] \oplus X_0^3[0]$ constant. The bit indices to be guessed are presented in Table 6.

Table 7. Indices of additional guess bits of X_0^1

$S_{T_4}(0)$	-
$S_{T_5}(5)$	18, 29, 38, 40, 60
$S_{T_6}(4)$	4, 13, 23, 59
$S_{T_7}(0)$	-
$S_{T_8}(1)$	17
$S_{T_9}(1)$	17
$S_{T_5} \cup S_{T_6} \cup S_{T_8} \cup S_{T_9}(10)$	4, 13, 17, 18, 23, 29, 38, 40, 59, 60

Therefore, we can make $Y_1^3[0]$ a constant by guessing 36 bits. The sets of guess bits that make $Y_1^3[3]$ and $Y_1^3[25]$ constants can also be obtained through simple rotation, each consisting of 36 bits. The union of these three sets of guess bits becomes the set of guess bits for making $X_1^4[0]$ a constant, and the size of the set is 57. Accordingly, we obtain the following property.

Property 4. For Ascon's permutation, when $X_1^1, X_2^1, X_3^1, X_4^1$ and the round constants are set to 0, 57 bits of X_0^1 must be guessed to make $X_1^4[j]$ a constant.

Now we turn to linearizing $Y_0^4[j]$ by observing X_0^4, X_2^4, X_3^4 and X_4^4. This proceeds similarly to the process of deriving Property 4.

$$X_0^4[0] = \underbrace{Y_0^3[0]}_{Const} \oplus Y_0^3[45] \oplus Y_0^3[36] \tag{16}$$

Here, $Y_0^3[0]$ becomes a constant by the process of deriving Property 4. If we analyze the guess bits required to linearize $Y_0^3[45]$, we can also find that for $Y_0^3[36]$ as well. The four terms X_0^4, X_2^4, X_3^4 and X_4^4 are expanded as follows, where the bit indices of X_0^1 required to make the term constant are represented under each term.

$$\begin{aligned}
X_0^4[0] &= \underbrace{Y_0^3[0]}_{Const} \oplus Y_0^3[45] \oplus Y_0^3[36] \\
&= C \oplus \underset{\mathbf{10}}{X_4^3[45]}X_1^3[45] \oplus X_3^3[45] \oplus X_2^3[45]X_1^3[45] \oplus X_2^3[45] \oplus X_1^3[45]X_0^3[45] \oplus X_1^3[45] \oplus X_0^3[45] \\
&\oplus \underset{\mathbf{10,12}}{X_4^3[36]}X_1^3[36] \oplus X_3^3[36] \oplus X_2^3[36]X_1^3[36] \oplus X_2^3[36] \oplus X_1^3[36]X_0^3[36] \oplus X_1^3[36] \oplus X_0^3[36] \\
X_2^4[0] &= \underbrace{Y_2^3[0]}_{Const} \oplus Y_2^3[63] \oplus Y_2^3[58] \\
&= C \oplus \underset{\mathbf{12,14,34,46,56}}{X_4^3[63]X_3^3[63]} \oplus \underset{\mathbf{12,46,56}}{X_4^3[63]} \oplus X_2^3[63] \oplus \underset{\mathbf{14,46,56}}{X_1^3[63]} \oplus 1 \\
&\oplus \underset{\mathbf{12,34}}{X_4^3[58]}X_3^3[58] \oplus \underset{\mathbf{12,34}}{X_4^3[58]} \oplus \underset{\mathbf{52}}{X_2^3[58]} \oplus X_1^3[58] \oplus 1 \\
X_3^4[0] &= \underbrace{Y_3^3[0]}_{Const} \oplus Y_3^3[54] \oplus Y_3^3[47] \\
&= C \oplus X_4^3[54]X_0^3[54] \oplus X_4^3[54] \oplus \underset{\mathbf{34}}{X_3^3[54]}X_0^3[54] \oplus \underset{\mathbf{34}}{X_3^3[54]} \oplus \underset{\mathbf{14,56}}{X_2^3[54]} \oplus X_1^3[54] \oplus X_0^3[54] \\
&\oplus \underset{\mathbf{12}}{X_4^3[47]}X_0^3[47] \oplus \underset{\mathbf{12}}{X_4^3[47]} \oplus X_3^3[47]X_0^3[47] \oplus X_3^3[47] \oplus \underset{\mathbf{46}}{X_2^3[47]} \oplus X_1^3[47] \oplus X_0^3[47] \\
X_4^4[0] &= \underbrace{Y_4^3[0]}_{Const} \oplus Y_4^3[57] \oplus Y_4^3[23] \\
&= C \oplus \underset{\mathbf{52}}{X_4^3[57]}X_1^3[57] \oplus \underset{\mathbf{52}}{X_4^3[57]} \oplus \underset{\mathbf{12}}{X_3^3[57]} \oplus X_1^3[57]\underset{\mathbf{10,12}}{X_0^3[57]} \oplus X_1^3[57] \\
&\oplus \underset{\mathbf{46,52}}{X_4^3[23]}X_1^3[23] \oplus \underset{\mathbf{46,52}}{X_4^3[23]} \oplus X_3^3[23] \oplus X_1^3[23]X_0^3[23] \oplus X_1^3[57]
\end{aligned} \tag{17}$$

If we guess the 12, 46, and 56 bit indices, we can linearize $Y_0^4[0]$ in Equation 9. Overall, we can linearize one bit $Y_0^4[0]$ by guessing 57 bits in Property 4 and an additional 3 bits. Finally, we get the following property.

Property 5. For ASCON's permutation, when $X_1^1, X_2^1, X_3^1, X_4^1$ and the round constants are set to 0, 60 bits of X_0^1 must be guessed to linearize one bit of Y_0^4.

Linearizing the two bits of the output requires at least 63 bit guesses. Therefore, with 60 bits guesses for linearizing one bit and 3 bits random guesses, we can expect to find a one-block solution for the preimage. However, we do not claim that 4-round ASCON-XOF with a 64-bit output is broken by our attack, as the advantage of our attack is relatively small.

5 Conclusion and Discussion

In this paper, we proposed security analysis for ASCON-XOF of the ASCON family, which is the final selection algorithm of the NIST LWC. We improved the complexity of the preimage attack on 2-round ASCON-XOF proposed by ASCON's designers by a factor of 2^5, and we developed this attack tool. Furthermore, we demonstrated that ASCON's permutation can be linearized up to 4 rounds, and the preimages of 3- and 4-round ASCON-XOF can be found faster than the generic complexity of 2^{64}. Our attacks primarily focus on analyzing the reduced-round ASCON-XOF without initialization, with both IV and round constants set to 0, and in some of the versions we attacked, actual round constants are considered. In the future work, it would be interesting to investigate optimal selection sets of guess bits by using automated tools, such as MILP and SAT, or ideas presented in previous results, such as [23].

Our main idea is to linearize the leaked polynomials from the hash value and then perform Gauss–Jordan elimination on them to determine the unknown variables. Despite the internal diffusion within ASCON's permutation, we demonstrate that it is possible to linearize one output bit by guessing 60 bits of the input, even after just 2 rounds of the function, and also up to 4 rounds. This also shows that it is impossible to linearize one bit of the output in this way by guessing less than 64 bits in more rounds than 4. Our results are important as they reveal the level of linearization of ASCON's permutation proposed by ASCON's designers in more depth and lead them to preimage attacks. Although our attacks cannot be extended to the full version of the cipher, we believe that our results will provide valuable insights into ASCON's security.

Acknowledgements. This work was supported as part of Military Crypto Research Center(UD210027XD) funded by Defense Acquisition Program Administration(DAPA) and Agency for Defense Development(ADD).

References

1. Bar-On, A., Dunkelman, O., Keller, N., Weizman, A.: Dlct: a new tool for differential-linear cryptanalysis. In: EUROCRYPT 2019. pp. 313–342. Springer (2019), https://doi.org/10.1007/978-3-030-17653-2_11
2. Bernstein, Daniel, J.: Second preimages for 6 (7 (8??)) rounds of keccak? Posted on the NIST mailing list (2010), https://ehash.iaik.tugraz.at/uploads/6/65/NIST-mailing-list_Bernstein-Daemen.txt
3. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: single-pass authenticated encryption and other applications. In: SAC 2011. pp. 320–337. Springer (2011), https://doi.org/10.1007/978-3-642-28496-0_19
4. Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Sponge functions. In: ECRYPT hash workshop. Citeseer (2007), https://csrc.nist.rip/groups/ST/hash/documents/JoanDaemen.pdf
5. Civek, A.B., Tezcan, C.: Differential-linear attacks on permutation ciphers revisited: Experiments on ascon and drygascon. In: ICISSP 2022. pp. 202–209. SCITEPRESS (2022), https://doi.org/10.5220/0010982600003120
6. Dobraunig, C., Eichlseder, M., Mangard, S., Mendel, F., Mennink, B., Primas, R., Unterluggauer, T.: Isap. Submission as a Finalist to the NIST Lightweight Crypto Standardization Process (2021), https://csrc.nist.gov/Projects/lightweight-cryptography/finalists
7. Dobraunig, C., Eichlseder, M., Mendel, F.: Heuristic tool for linear cryptanalysis with applications to caesar candidates. In: ASIACRYPT 2015. pp. 490–509. Springer (2015), https://doi.org/10.1007/978-3-662-48800-3_20
8. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Cryptanalysis of ascon. In: CT-RSA 2015. pp. 371–387. Springer (2015), https://doi.org/10.1007/978-3-319-16715-2_20
9. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2. Submission to Round 3 of the CAESAR competition (2016), https://competitions.cr.yp.to/round3/asconv12.pdf
10. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Preliminary analysis of ascon-xof and ascon-hash. Technique Report (2019), https://ascon.iaik.tugraz.at/files/Preliminary_Analysis_of_Ascon-Xof_and_Ascon-Hash_v01.pdf
11. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2: Lightweight authenticated encryption and hashing. J. Cryptol. **34**(3), 33 (2021), https://doi.org/10.1007/s00145-021-09398-9
12. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1.2 submission to nist. LWC Final round submission (2021), https://csrc.nist.gov/Projects/lightweight-cryptography/finalists
13. Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon resources (Accessed Oct 2022), https://ascon.iaik.tugraz.at/resources.html
14. Dwivedi, A.D., Klouček, M., Morawiecki, P., Nikolic, I., Pieprzyk, J., Wöjtowicz, S.: Sat-based cryptanalysis of authenticated ciphers from the caesar competition. ICETE 2017 pp. 237–246 (2017), https://doi.org/10.5220/0006387302370246
15. Dworkin, M.: Sha-3 standard: Permutation-based hash and extendable-output functions (2015), https://doi.org/10.6028/NIST.FIPS.202
16. Dworkin, M., Feldman, L., Witte, G.: Additional secure hash algorithm standards offer new opportunities for data protection (2015), https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=919417

17. Erlacher, J., Mendel, F., Eichlseder, M.: Bounds for the security of ascon against differential and linear cryptanalysis. IACR Trans. Symmetric Cryptol. **2022**(1), 64–87 (2022), https://doi.org/10.46586/tosc.v2022.i1.64-87
18. Gerault, D., Peyrin, T., Tan, Q.Q.: Exploring differential-based distinguishers and forgeries for ascon. IACR Trans. Symmetric Cryptol. **2021**(3), 102–136 (2021), https://doi.org/10.46586/tosc.v2021.i3.102-136
19. Göloğlu, F., Rijmen, V., Wang, Q.: On the division property of s-boxes. Cryptology ePrint Archive (2016), http://eprint.iacr.org/2016/188
20. Jovanovic, P., Luykx, A., Mennink, B.: Beyond 2 c/2 security in sponge-based authenticated encryption modes. In: ASIACRYPT 2014. pp. 85–104. Springer (2014), https://doi.org/10.1007/978-3-662-45611-8_5
21. Kelsey, J., Chang, S.j., Perlner, R.: Sha-3 derived functions: cshake, kmac, tuplehash, and parallelhash. NIST special publication **800**, 185 (2016), https://www.nist.gov/publications/sha-3-derived-functions-cshake-kmac-tuplehash-and-parallelhash
22. Leander, G., Tezcan, C., Wiemer, F.: Searching for subspace trails and truncated differentials **2018**(1), 74–100 (2018), https://doi.org/10.13154/tosc.v2018.i1.74-100
23. Li, H., He, L., Chen, S., Guo, J., Qiu, W.: Automatic preimage attack framework on ascon using a linearize-and-guess approach. IACR Trans. Symmetric Cryptol. **2023**(3), 74–100 (2023), https://tosc.iacr.org/index.php/ToSC/article/view/11185
24. Li, Y., Zhang, G., Wang, W., Wang, M.: Cryptanalysis of round-reduced ascon. Science China Information Sciences **60**(3), 1–2 (2017), https://doi.org/10.1007/s11432-016-0283-3
25. Li, Z., Dong, X., Wang, X.: Conditional cube attack on round-reduced ascon. IACR Trans. Symmetric Cryptol. **2017**(1), 175–202 (2017), https://doi.org/10.13154/tosc.v2017.i1.175-202
26. Liu, M., Lu, X., Lin, D.: Differential-linear cryptanalysis from an algebraic perspective. In: CRYPTO 2021. pp. 247–277. Springer (2021), https://doi.org/10.1007/978-3-030-84252-9_9
27. Makarim, R.H., Rohit, R.: Towards tight differential bounds of ascon: A hybrid usage of smt and milp. IACR Trans. Symmetric Cryptol. **2022**(3), 303–340 (2022), https://doi.org/10.46586/tosc.v2022.i3.303-340
28. NIST: Submission requirements and evaluation criteria for the lightweight cryptography standardization process (2018), https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf
29. Qin, L., Hua, J., Dong, X., Yan, H., Wang, X.: Meet-in-the-middle preimage attacks on sponge-based hashing. In: EUROCRYPT 2023. Lecture Notes in Computer Science, vol. 14007, pp. 158–188. Springer (2023), https://doi.org/10.1007/978-3-031-30634-1_6
30. Qin, L., Zhao, B., Hua, J., Dong, X., Wang, X.: Weak-diffusion structure: Meet-in-the-middle attacks on sponge-based hashing revisited. IACR Cryptol. ePrint Arch. p. 518 (2023), https://eprint.iacr.org/2023/518
31. Rohit, R., Hu, K., Sarkar, S., Sun, S.: Misuse-free key-recovery and distinguishing attacks on 7-round ascon. IACR Trans. Symmetric Cryptol. **2021**(1), 130–155 (2021), https://doi.org/10.46586/tosc.v2021.i1.130-155
32. 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: ASIACRYPT 2014. Lecture Notes in Computer Science, vol. 8873, pp. 158–178. Springer (2014), https://doi.org/10.1007/978-3-662-45611-8_9
33. Todo, Y.: Structural evaluation by generalized integral property. In: EUROCRYPT 2015. pp. 287–314. Springer (2015), https://doi.org/10.1007/978-3-662-46800-5_12
34. Weatherley, R.: Additional modes for lwc finalists technical report, version 1.0 (2021), https://rweather.github.io/lwc-finalists/lwc-modes-v1-0.pdf
35. Wiethuechter, A., Card, S.W., Moskowitz, R.: DRIP Entity Tag Authentication Formats & Protocols for Broadcast Remote ID. Internet-Draft draft-ietf-drip-auth-29, Internet Engineering Task Force (Feb 2023), https://datatracker.ietf.org/doc/draft-ietf-drip-auth/29/, work in Progress
36. Yan, H., Lai, X., Wang, L., Yu, Y., Xing, Y.: New zero-sum distinguishers on full 24-round keccak-f using the division property. IET Information Security **13**(5), 469–478 (2019), https://doi.org/10.1049/iet-ifs.2018.5263
37. Zong, R., Dong, X., Wang, X.: Collision attacks on round-reduced gimli-hash/ascon-xof/ascon-hash. IACR Cryptol. ePrint Arch. p. 1115 (2019), https://eprint.iacr.org/2019/1115,