IACR Transactions on Symmetric Cryptology
ISSN 2519-173X, Vol. 2022, No. 4, pp. 120–144. DOI:10.46586/tosc.v2022.i4.120-144

Practical Cube Attack against Nonce-Misused Ascon

Jules Baudrin, Anne Canteaut and Léo Perrin

Inria, Paris, France
{jules.baudrin,anne.canteaut,leo.perrin}@inria.fr

Abstract. Ascon is a sponge-based Authenticated Encryption with Associated Data that was selected as both one of the winners of the CAESAR competition and one of the finalists of the NIST lightweight cryptography standardization effort. As this competition comes to an end, we analyse the security of this algorithm against cube attacks. We present a practical cube attack against the full 6-round encryption in Ascon in the nonce-misuse setting. We note right away that this attack does not violate the security claims made by the designers of Ascon, due to this setting.

Our cryptanalysis is a conditional cube attack that is capable of recovering the full capacity in practical time; but for Ascon-128, its extension to a key recovery or a forgery is still an open question. First, a careful analysis of the maximum-degree terms in the algebraic normal form of the Ascon permutation allows us to derive linear equations in half of the capacity bits given enough cube sums of dimension 32. Then, depending on the results of this first phase, we identify smaller-degree cubes that allow us to recover the remaining half of the capacity. Overall, our cryptanalysis has a complexity of about 2^{40} adaptatively chosen plaintexts, and about 2^{40} calls to the permutation. We have implemented the full attack and our experiments confirm our claims.

Our results are built on a theoretical framework which allows us to easily identify monomials whose cube-sums provide linear equations in the capacity bits. The coefficients of these monomials have a more general form than those used in the previous attacks against Ascon, and our method enables us to re-frame previous results in a simpler form. Overall, it enables to gain a deeper understanding of the properties of the permutation, and in particular of its S-box, that make such state-recoveries possible.

Keywords: Ascon · cube attack · algebraic attack · lightweight cryptography · CAESAR · nonce-misuse

1 Introduction

As the NIST lightweight cryptography standardization effort reaches its final stage, it is crucial to investigate the security of the ten finalists. In this paper, we focus our attention on Ascon, a family of lightweight primitives. It is also part of the final portfolio of the CAESAR competition in the "lightweight applications" category [CAE14].

We focus here on the primary Authenticated Encryption with Associated Data mode (AEAD [Rog02]) of Ascon, namely Ascon-128, which motivates the attack model we choose. In this mode, Ascon aims to provide integrity and confidentiality both with authenticity in an effective integrated manner [BN00]. On top of that, it ensures the authenticity of associated public data (such as public headers). Since our attack recovers the whole internal state of the cipher (which follows the sponge construction) after the initialization, it obviously applies to Ascon-80pq. Indeed, these two variants differ only from their initialization phases.

Received: 2022-06-01 Revised: 2022-09-01 Accepted: 2022-11-01 Published: 2022-12-07

Due to the low degree of the round function in Ascon, so-called *cube attacks* and other cryptanalysis directions related to higher-order differentials are tempting attack vectors. Following several works investigating such attacks against Ascon, we add new evidence that this intuition is correct. We focus on the terms of highest degree in the Algebraic Normal Form (see below) of the inner permutation of Ascon, and we identify strong and (until now) unknown patterns that allow us to mount a practical capacity-recovery attack against the full primitive. This result comes however with a strong caveat: it assumes significant nonce-misuse, a context in which the designers of Ascon have not made any security claim. Because of the keyed initialization and finalization, it is unclear how such a state-recovery could lead to a future key-recovery or forgery. Thus, our results *do not* violate their security claims.

Our paper is structured as follows. First, Section 2 presents the necessary background. Our attack model is then described in Section 3. Section 4 presents the main properties of high-degree monomials used to mount our recovery attack, which is detailed in Section 5. In Section 6, possible counter-measures are analyzed. Section 7 concludes the paper. It is worth noting that an independent team of cryptographers analyzed Ascon and obtained results that have some overlap with ours [CHK22]. The main differences between our attack and this concurrent work are then summarized at the end of Section 5.

2 Preliminaries

2.1 Notation

Let $\mathbb{F}_2$ denote the finite field with two elements, while $\mathbb{F}_2^n$ denotes the vector space of dimension n over $\mathbb{F}_2$. The bitwise addition (XOR) of two binary vectors is denoted by $\oplus$ while $||$ is used to indicate the concatenation of two bitstrings. Given two vectors $u, v \in \mathbb{F}_2^n$, $u \preccurlyeq v$ stands for $u_i \leq v_i, \forall\ i \in \{0, \dots, n-1\}$. **Supp** stands for support, while **wt** stands for Hamming weight; in others words, given a vector of $\mathbb{F}_2^n$, $u = (u_0, \cdots, u_{n-1})$, $\mathbf{Supp}(u)$ is defined as $\mathbf{Supp}(u) := \{i, u_i = 1\}$ and $\mathbf{wt}(u)$ is defined as $\mathbf{wt}(u) := |\mathbf{Supp}(u)|$.

Boolean Functions. Given a vector $u \in \mathbb{F}_2^n$ and a family of n variables $x_0, \cdots, x_{n-1}$ we denote x^u the monomial

$$x^u := \prod_{i=0}^{n-1} x_i^{u_i} .$$

Any Boolean function $f\colon \mathbb{F}_2^n \to \mathbb{F}_2$ can be uniquely represented by its *Algebraic Normal Form* (or ANF) which is a polynomial in $\mathbb{F}_2[x_0, \cdots, x_{n-1}]/(x_0^2 + x_0, \cdots, x_{n-1}^2 + x_{n-1})$:

$$f(x) = \sum_{u \in \mathbb{F}_2^n} \alpha_u x^u \text{ with } \alpha_u \in \mathbb{F}_2 .$$

The ANF of a vectorial Boolean function is the list of ANFs of its coordinate functions.

For any $u \in \mathbb{F}_2^n$, the coefficient α_u of the monomial x^u can be computed from the values of the function by the Möbius transform:

$$\alpha_u = \sum_{x \preccurlyeq u} f(x). \tag{1}$$

Keyed Boolean Functions. It is often necessary to distinguish controllable *public variables* from inaccessible *secret variables.* We denote public variables x_i, and use different variable names for the secret variables (such as k_i for key variables, or a_i, b_i, c_i, d_i for capacity variables) which will be clarified when used. With such a distinction, when a Boolean function depends on m public variables and n key variables, we look at it as

$$f = \sum_{u \in \mathbb{F}_2^m} \alpha_u x^u,$$

where each α_u lies in $\mathbb{F}_2[k_0, \cdots, k_{n-1}]/(k_0^2 + k_0, \cdots, k_{n-1}^2 + k_{n-1})$. For a given $u \in \mathbb{F}_2^n$, α_u is referred as the *coefficient of* x^u. When referring to the degree of a Boolean function, we mean the degree in public variables: $\deg(f) := \max\{\mathbf{wt}(u), \alpha_u \neq 0\}$. Terms of degree $\deg(f)$ are referred as *highest-degree terms* and terms of degree $(\deg(f) - 1)$ as *sub-leading terms.* It is worth noting that, when x^u is a highest-degree monomial of f, its coefficient α_u can also be referred as the *superpoly* of x^u in f, according to the terminology introduced by Dinur & Shamir [DS09]. However, since we do not restrict ourselves to highest-degree monomials, we rather use the *coefficient* denomination.

Equation (1) can then be adapted to recover the value of a given coefficient α_u.

Proposition 1 (Computation of a coefficient of a keyed Boolean function)**.** *Let f be a Boolean function of m public variables $(x_0, \cdots, x_{m-1})$ and n secret variables $(k_0, \cdots, k_{n-1})$. Let $\sum_{u \in \mathbb{F}_2^m} \alpha_u x^u$ denote its ANF, where each α_u lies in $\mathbb{F}_2[k_0, \cdots, k_{n-1}]/(k_0^2 + k_0, \cdots, k_{n-1}^2 + k_{n-1})$. Then, for any $u \in \mathbb{F}_2^m$, the coefficient α_u satisfies:*

$$\alpha_u = \sum_{x \preccurlyeq u} f(x, k_0, \cdots, k_{n-1}).$$

The Cube Notation. In Proposition 1, the set of vectors x such that $x \preccurlyeq u$ is a linear subspace of dimension $\mathbf{wt}(u)$, spanned by $\{\beta^i, i \in \mathbf{Supp}(u)\}$, where β^i denotes the i-th vector of the canonical basis, i.e., $\beta^i = (\delta_{i,j})_{j \in \{0,\ldots,n-1\}}$, with $\delta_{i,i} = 1$ and $\delta_{i,j} = 0$ for all $i, j \in \{0, \ldots, n-1\}, i \neq j$. Because of this point of view, the subspace is usually referred as a *cube of dimension* $\mathbf{wt}(u)$ and the summation process referred as a *cube-sum*; hence the name *cube attack.*

In this work, we only apply cube-sums over canonically-aligned subspaces. Therefore, there are one-to-one correspondences between the monomial x^u, the set $\mathbf{Supp}(u)$ and the linear subspace $\mathrm{Span}(\{\beta^i, i \in \mathbf{Supp}(u)\})$. Because of those identifications, selecting a monomial, a set of indices or a cube will refer to the same selection process; the ultimate goal being the computation of the corresponding coefficient α_u.

2.2 Ascon

The design of Ascon is based on the permutation-oriented Sponge Duplex mode of operation [BDPV12, BDPV11a]. Figure 1 presents the AEAD encryption mode. Most notably, $\Sigma_{\mathrm{AD}}, \Sigma_{\mathrm{E}}, \Sigma_{\mathrm{F}}$ respectively denote the 320-bit state before the processing of associated data, before encryption and before finalization.

The designers thus mainly focused on the design of the permutation p which also inherits from other already-standardized designs, such as the permutation of SHA-3 [BDPV11b].

The claimed level of security for Ascon-128 is 128 bits in terms of plaintext confidentiality and plaintext/data/nonce integrity under three hypotheses: the single usage of each nonce, the outputting of a decrypted plaintext only if the tag is correct, and the encryption of less than 2^{64} blocks using the same key.

The Permutation. The permutation p is the core element of the design. It operates on a 320-bit state (usually decomposed into five 64-bit words, $S = X_0||X_1||X_2||X_3||X_4$). It is built as a Substitution Permutation Network (SPN): p is the composition of a

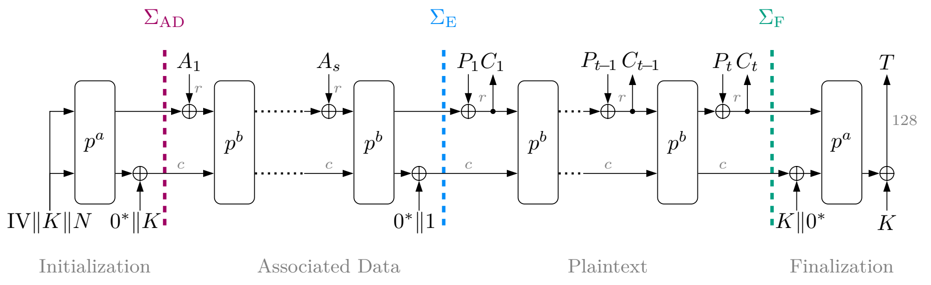

Figure 1: ASCON AEAD encryption.[1]

constant addition, a non-linear substitution layer and a linear diffusion transformation: $p = p_L \circ p_S \circ p_C$.

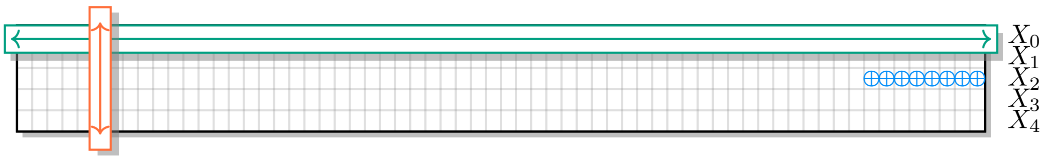

Figure 2: The column-wise S-box layer, the row-wise linear layer and the constant addition.

The constant adding step consists in XORing an 8-bit constant to the word X_2. The constant only depends on the index of the current iteration. All involved constants are then based on incremented and decremented counters, thus easily computable.

The substitution layer is made of 64 parallel calls to a single Substitution-box (S-box) on each column of the state. The S-box used in ASCON is a quadratic 5-bit permutation. Figure 3 presents the algebraic normal form of the S-box in ASCON. The low algebraic degree of the S-box (it is quadratic) plays a crucial role in the previous cube attacks against ASCON (see Section 2.3). It is also the case in the attack we present in Section 5.

The linear diffusion layer is made of five calls to five different linear functions Σ_i on each row of the state. For each $i \in \{0, \dots, 4\}$, $\Sigma_i(X_i)$ is built as the XOR of the i-th row X_i with two rotated versions of itself. The indices of rotations are fixed and only depend on i.

$$\begin{aligned}
y_0 &= x_4x_1 + x_3 + x_2x_1 + x_2 + x_1x_0 + x_1 + x_0 & \Sigma_0(X_0) &= X_0 \oplus (X_0 \ggg 19) \oplus (X_0 \ggg 28)\\
y_1 &= x_4 + x_3x_2 + x_3x_1 + x_3 + x_2x_1 + x_2 + x_1 + x_0 & \Sigma_1(X_1) &= X_1 \oplus (X_1 \ggg 61) \oplus (X_1 \ggg 39)\\
y_2 &= x_4x_3 + x_4 + x_2 + x_1 + 1 & \Sigma_2(X_2) &= X_2 \oplus (X_2 \ggg 1) \oplus (X_2 \ggg 6)\\
y_3 &= x_4x_0 + x_4 + x_3x_0 + x_3 + x_2 + x_1 + x_0 & \Sigma_3(X_3) &= X_3 \oplus (X_3 \ggg 10) \oplus (X_3 \ggg 17)\\
y_4 &= x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 & \Sigma_4(X_4) &= X_4 \oplus (X_4 \ggg 7) \oplus (X_4 \ggg 41)
\end{aligned}$$

Figure 3: ANF of the S-box (left) and the linear layer (right) of ASCON.

The iterated permutation p^6. In the nonce-misuse setting, the main object we focus on is the iterated permutation p^6. Because of its iterated form and the composed structure of p, this study is done round by round, or layer by layer, as depicted in Figure 4.

[1]All ASCON figures in this document are highly based on (if not identical to) the ones found on the page of the designers [DEMS] and on the TikZ for Cryptographers repository [Jea16].

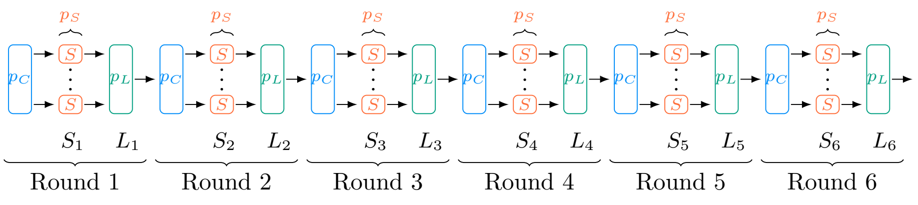
p_S
p_C
p_L
S
S_1 L_1 S_2 L_2 S_3 L_3 S_4 L_4 S_5 L_5 S_6 L_6
Round 1
Round 2
Round 3
Round 4
Round 5
Round 6

Figure 4: The iterated permutation p^6.

2.3 Related Work

With the point of view introduced in Section 2.1, it is possible to study the ANF of a symmetric cipher "coefficient-wise". In the case of Ascon, it already led to distinguishing attacks (see Table 7 in Appendix A). We are here interested in using this approach to mount recovery attacks, often named *cube attacks* (see Table 1).

Table 1: Summary of cube-like key-recoveries and state-recoveries against Ascon.

Attack type	Target	Nb of rounds	Data / Time	Method	Source
Nonce-respecting key-recovery	Initialization	5/12	$2^{19}/2^{35}$	Cube	[DEMS15]
		6/12	$2^{34}/2^{66}$	Cube	[DEMS15]
		5/12	2^{24}	Cond. cube	[LDW17]
		6/12	2^{40}	Cond. cube	[LDW17]
		7/12	$2^{77.2}$ / $2^{103.9}$	Cond. cube	[LDW17]
		7/12	$2^{77.2}$ / 2^{77}	Cond. cube†	[LDW17]
		7/12	2^{64} / 2^{123}	Cube	[RHSS21]
		7/12	2^{64} / 2^{97}	Cube†	[RS21]
		7/12	2^{63} / $2^{115.2}$	Cube†	[RS21]
Nonce-misuse key-recovery	Initialization	7/12	? / 2^{97}	Cube-like	[LZWW17]
Nonce-misuse state-recovery	Encryption	5/6	? / 2^{66}	Cube-like	[LZWW17]
		6/6	$2^{44.8}/2^{128}$	Cond. cube	[CHK22]
		6/6	$\leq 2^{40}$	**Cond. cube**	**Sec. 5**

† stands for "Weak-key subspace", Cond. for conditional.

Cube attacks were introduced by Dinur & Shamir [DS09] in order to recover the values of some secret variables. More precisely, the goal of the attack is to obtain, by the use of chosen *cubes*, enough linear equations in secret variables that it becomes possible to solve the resulting system. The attack is thus composed of two stages:

1. Firstly, the offline phase focuses on coefficients associated to given monomials in the ANF of some fixed well-chosen output coordinates, which are (almost surely) known to be linear in the secret variables; from either linearity tests or theoretical arguments. Then, thanks to a complete offline access to the primitive (both public and secret variables can be set), the linear expressions can be recovered by interpolation.

2. Secondly, the online phase computes the actual values of the previously-targeted coefficients by querying the online oracle (*i.e.*, a limited-access blackbox where only public variables can be set). Thanks to the Möbius transform (see Proposition 1), the value of any coefficient in the ANF can be recovered in this setting. Finally, with

pairs of linear equations/values, an adversary can solve the system and recover (part of) the unknown bits.

Several cube-like attacks have then been applied to round-reduced variants of the Ascon initialization in order to recover some key bits. They adapted the original cube attack since focusing on coefficients which are linear in the key bits is too restrictive. First, the designers of Ascon [DEMS15] mounted a cube-attack on 5 rounds of the initialization by targeting coefficients which only depend on a small number of key bits. This enables them to recover the whole key in a divide-and-conquer manner.

Li *et al.* [LDW17] continued analyzing the resistance of Ascon against cube-like attacks by adapting and generalizing the conditional cube attacks against Keccak introduced by Huang *et al.* [HWX$^+$17]. They searched for monomials whose coefficients in all output coordinates share a linear divisor. For 5 and 6 rounds, they were able to exhibit such monomials without determining the entire expression of the coefficients. If such a common divisor exists, an adversary is able to deduce its value from the value of the cube sum, as the linear factor influences all bits in the cube-sum vector. This 5/6-round attack has been extended to 7 rounds (out of 12) by replacing the common linear divisor by a family of linear polynomials in which lies at least one divisor for each coefficient. Then, Rohit *et al.* [RHSS21] presented the first 7-round misuse-free key-recovery cube-attack on Ascon, which does not exceed the limitation of 2^{64} encrypted data blocks. In [RS21], this 7-round attack is adapted to the particular case of weak-key spaces and the data complexity is improved.

Remark 1. We would like to emphasize the difference between cube and conditional cube methods. Both methods were already applied to the Ascon initialization and their complexities greatly differ. It is particularly striking in Table 1, while comparing both methods in the case of initialization reduced to 5 or 6 rounds. Indeed, a conditional cube attack can be seen as an alternative to the costly offline phase of a “standard” cube attack. In a standard cube attack, an adversary first has to compute offline the table of values of the coefficient she will later target during the online phase. The computation of this table can be long and is proportional to the memory needed to store the table. However it only has to be done once. Afterwards, the online-time cost (which is proportional to the data complexity) is low. A conditional cube attack offers another trade-off: avoiding the precomputation is possible at the cost of higher data and online-time complexities.

All of the aforementioned works study the nonce-respecting scenarios and thus focus on the initialization. However, implementation errors will eventually happen and sometimes with high risk: we show in Section 5 that, if a nonce is reused many times, confidentiality is compromised.

3 Nonce-Misuse Setting and Attack Model

The nonce-misuse scenario assumes, *contrary to the recommendations of the designers*, that a key/nonce pair is reused many times to encrypt plaintexts. In this situation, the state after initialization, Σ_{AD} on Figure 1, can be considered fixed once and for all when encrypting several plaintexts. In the following, we focus on an adversary who interacts with the cipher by querying the encryption of chosen plaintexts having the following form: no associated data is processed, and the chosen plaintexts consist of two blocks of the form $(P_1, 0^*)$, where P_1 takes any 64-bit value and 0^* denotes the all-zero block.

In other words, P_1 is inserted in the first row (known as the *rate*, or the outer part) of the internal state after initialization. The adversary then gets the corresponding 2-block ciphertext. In our attack, she will omit the first ciphertext block and focuses on the second one C_2, as depicted on Figure 5.

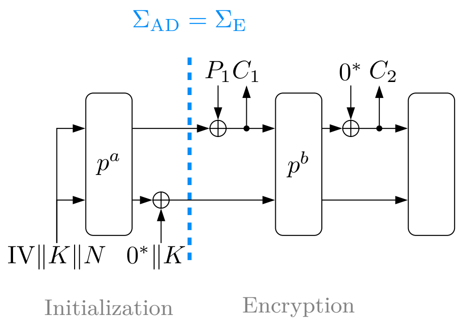

Figure 5: Nonce-misuse attack model.

In this nonce-misuse scenario, the goal of the adversary is to recover the *capacity* of the state Σ_E, that is, the unknown inner part of the state just before the encryption phase (see Figure 1). If an adversary manages to recover the inner part of Σ_E, then the full state Σ_E is recovered, the outer part of the state being easily recovered by computing $P_1 \oplus C_1$ with the first plaintext-ciphertext pair. From there, the full state Σ_{AD} can be immediately recovered because $\Sigma_E = \Sigma_{AD}$ (see Figure 5). For a fixed key/nonce pair, once Σ_{AD} is recovered, any full internal state can be recovered up to the state Σ_F, and thus any other message, encrypted under the same key/nonce pair, can be decrypted.

If *the same* associated data is processed before encrypting *all* required plaintexts (instead of no associated data), Σ_E can be recovered in the same way. In that case, the equality between Σ_{AD} and Σ_E does not necessarily hold, but Σ_{AD} can be directly deduced from the knowledge of Σ_E and the associated data, by inverting the data processing phase.

It is worth noting that our attack recovers the whole state Σ_{AD} as soon as enough plaintexts of the previous form are encrypted from the same internal state Σ_E, *i.e.*, from the same triple of key, nonce and associated data. This differs from the attack scenario in the concurrent work [CHK22] which recovers a part of the state Σ_E only if it satisfies a few conditions. It follows that the first cube-sum computation (called "Step 1") of the attack in [CHK22] needs to be repeated for 32 triples of key, nonce and associated data in average, until the corresponding state can be recovered.

Comparison with [VV18]. It has been shown in [VV18] that any cipher following the Sponge Duplex construction is vulnerable to a generic attack which recovers a given plaintext in the nonce-misuse scenario. This generic attack is very cheap: a single adapted query is necessary for each block that needs to be decrypted. It has then a lower query complexity than our attack for messages whose length does not exceed 2^{40} blocks. However, our attack and the generic attack differ both in their settings and their intention: contrary to [VV18], we present an inner-state-recovery which gives new insights about the permutation used in Ascon. Moreover, it breaks the confidentiality of the next blocks encrypted with the same key-nonce pair *without interacting with the keyed-encryption oracle anymore*. The higher cost is explained by this internal-state-recovery which is interesting in itself and might lead to other attacks.

Another important remark is that a state-recovery does not directly enable a key-recovery (for example by going through the initialization backward) or a tag forgery (by going through the finalization): the XOR with the key just before Σ_{AD}, and just after Σ_F, prevents such attacks. However in the case of Ascon-80pq *only*, as Chang *et al.* [CHK22] pointed out, a state-recovery as we describe in Section 5 can lead to a key-recovery of the 160-bit key in less than 2^{160} operations. Nevertheless, this requires a nonce-misuse setting, and more operations than claimed by the designers for the nonce-respecting setting (128 bits of security are claimed for nonce-respecting Ascon-80pq).

4 Properties of High-Degree Monomials in the Nonce-Misuse Scenario

Four rows of 64 bits in the state Σ_E need to be recovered. We name them, a, b, c, d from top to bottom. Moreover, a_i, with $i \in \{0, \dots, 63\}$, (resp. b_i, c_i, d_i) represents the ith bit of the first (resp. second, third and fourth) row of the capacity. We use x_i to represent the ith controlled input bit (*i.e.* the ith public variable). Using the point of view introduced in Section 2.1, we look at the ANF of a coordinate as a "keyed Boolean function":

$$f = \sum_{u \in \mathbb{F}_2^n} \alpha_u x^u, \text{ with } \alpha_u \in \mathbb{F}_2[a_0, \cdots, d_{63}]/(a_0^2 + a_0, \cdots, d_{63}^2 + d_{63}) .$$

Table 2: ANF of Column i after initialization (left) and after the first S-box layer (right).

x_i	$(a_i + 1)x_i$	$+$	$a_ib_i + a_id_i + a_i + b_i + c_i$
a_i	x_i	$+$	$a_ib_i + a_ic_i + b_ic_i + a_i + b_i + c_i + d_i$
b_i	0	$+$	$c_id_i + a_i + b_i + d_i + 1$
c_i	$(c_i + d_i + 1)x_i$	$+$	$a_i + b_i + c_i + d_i$
d_i	a_ix_i	$+$	$a_id_i + a_i + c_i + d_i$

4.1 General properties of highest-degree and sub-leading terms

While the previous cube-like attacks on Ascon initialization focus on specific monomials whose coefficients in all coordinates share a linear factor in the key bits, we now provide an in-depth analysis of high-degree terms which shows that a more general property can be exploited in the nonce-misuse scenario.

Proposition 2. *In the nonce-misuse scenario, the highest degree of a monomial x^u at round r, $r \in \{1, \dots, 6\}$ is 2^{r-1}. For each round r, $r \in \{1, \dots, 6\}$, this bound is tight for at least one key/nonce pair in at least one out of the 320 coordinates. Moreover, let us assume that a highest-degree monomial x^u appears in a coordinate at Round r, $r \in \{2, \dots, 6\}$,* i.e., *$\alpha_u \neq 0$. Then, it was obtained from one or more products of two highest-degree terms at Round $(r - 1)$. In other words, α_u can be seen as a sum, each term of the sum being a product of two coefficients of highest-degree terms one round before.*

Proposition 2 is a consequence of the fact that, in the first S-box layer, no public variable is multiplied with another public variable, as public variables are all inserted in the same row, while the S-box is applied column-wise. So after one round the highest-degree is still 1. Afterwards, the upper-bound doubles at each round because the S-box is quadratic. Some of the studied coefficients of monomials of degree 32 (see for example Section 4.2) are non-zero polynomials in the nonce and key variables. Thus, the upper-bound of 32 is tight for at least one key/nonce pair and for at least one coordinate after 6 rounds. This also implies that the upper-bound of 2^{r-1} is tight for at least one key/nonce pair and for at least 2^{6-r} coordinates after r rounds, $r \in \{1, \dots, 6\}$.

Remark 2. From our observations, it seems very likely that an even stronger result holds: the bound seems to be tight for every key/nonce pair and for every coordinate.

From Proposition 2, we deduce the following corollary.

Corollary 1. *Let α_u be the coefficient of x^u in any output coordinate after Round r, $r \in \{1, \ldots, 6\}$, where $\mathbf{wt}(u) = 2^{r-1}$. Then, α_u can be viewed as a sum of products of 2^{r-1} coefficients of degree-1 terms after one round. In other words, each product in this sum has the following structure:*

$$\prod_{i, u_i=1} \ell_i \text{ , where } \ell_i \in \{a_i \oplus 1,\ 1,\ c_i \oplus d_i \oplus 1,\ a_i\} \ .$$

Proof. The fact that α_u is a sum of products of 2^{r-1} coefficients of degree-1 terms after one round is derived from Proposition 2 by a simple induction reasoning. According to the ANF after the first S-box layer (given in Table 2), if x_i is present in a coordinate, then its coefficient is either $a_i \oplus 1,\ 1,\ c_i \oplus d_i \oplus 1$ or a_i depending on the row of the coordinate we consider. The linear layer, which is applied row-wise, preserves this property. Thus, if x_i is present in a coordinate after one round, then its coefficient is either $a_i \oplus 1,\ 1,\ c_i \oplus d_i \oplus 1$ or a_i. Since each product appearing in α_u is the product of 32 coefficients of monomials of degree 1 after one round, we deduce that it has the aforementioned structure. $\square$

In the following, we denote $e_i := c_i \oplus d_i \oplus 1$ for any $i \in \{0, \ldots, 63\}$. Then, targeting highest-degree monomials and their coefficients can only enable the recovery of the bits of a and $e := c \oplus d$. If we want to recover b, c or d, *sub-leading terms* can be used.

We study those using the following proposition (and its corollary).

Proposition 3. *In the nonce-misuse scenario, a sub-leading monomial at Round r, $r \in \{1, \ldots, 6\}$ has degree $2^{r-1} - 1$. Moreover, any sub-leading monomial x^u appearing in a coordinate with coefficient $\alpha_u \neq 0$, is obtained from one or more products of one of the following forms:*

- *a product of two highest-degree terms at Round $(r-1)$ sharing a public variable as common divisor, or*
- *a product of a highest-degree term and a sub-leading term at Round $(r-1)$.*

Therefore, α_u can be seen as a sum, where each term in the sum is either a product of two coefficients of highest-degree terms or a product of one coefficient of highest-degree term by one coefficient of a sub-leading term one round before.

Corollary 2. *Let α_u be the coefficient of x^u in any output coordinate after Round r, $r \in \{2, \ldots, 6\}$, with $\mathbf{wt}(u) = 2^{r-1} - 1$. Then, α_u can be viewed as a sum of products of 2^{r-1} coefficients of terms after the second constant addition,*[2] *each product having one of the following forms:*

- *a product of 2^{r-1} coefficients of terms of degree 1 (among the 2^{r-1} monomials of degree 1, two are identical, but the two respective coefficients may differ), or*
- *a product of one constant term and $(2^{r-1} - 1)$ coefficients of terms of degree 1.*

From there, because the coefficients of highest-degree and sub-leading terms only depend on the coefficients of highest-degree and sub-leading terms after the first linear layer or after the second constant addition, we observe that only the first two constant additions can influence them; the others can thus be omitted. The first addition occurs just after initialization, flipping four unknown bits. Recovering those bits or their flipped values being equivalent, we can omit the first addition (at the (null) cost of flipping four recovered bits at the end of the attack). Therefore, the situation for highest-degree terms is completely cyclic, as stated in Proposition 4.

[2]Four constant terms (after the first linear layer) are flipped by the second constant addition.

Proposition 4 (Rotation invariance)**.** *Let us consider the nonce-misuse scenario with the first constant addition being omitted. Let us assume that $\alpha_{u,i,j}x^u$ appears in the ANF of the j-th output coordinate of Row i at Round r, with $\mathbf{wt}(u) = 2^{r-1}$. Let $k \in \{0, \dots, 63\}$ and let $u^k \in \mathbb{F}_2^n$ be defined by $\mathbf{Supp}(u^k) = k + \mathbf{Supp}(u)$, where the addition is computed modulo 64.*

Then, the monomial x^{u^k} appears in the ANF of the $(j+k)$-th[3] output coordinate of Row i at Round r. Moreover, its coefficient $\alpha_{u^k,i,j+k}$ is built, based on $\alpha_{u,i,j}$, by shifting all the indices of the variables by k modulo 64. More precisely, for $s \in (\mathbb{F}_2^{64})^4$, if the monomial $a^{s_0}b^{s_1}c^{s_2}d^{s_3}$ appears in the expression of $\alpha_{u,i,j}$, then $a^{s_0^k}b^{s_1^k}c^{s_2^k}d^{s_3^k}$ appears in the expression of $\alpha_{u^k,i,j+k}$, where s_ℓ^k is defined by $\mathbf{Supp}(s_\ell^k) = k + \mathbf{Supp}(s_\ell)$ for $\ell \in \{0, \dots, 3\}$.

Finally, the second constant addition flips four bits after the first linear layer. It can thus modify four constant terms at the beginning of Round 2 and then, according to Corollary 2, it may influence the coefficients of sub-leading terms in the following rounds. However, it is easy to keep track of this influence, as we will see later.

4.2 Two particular classes of highest-degree terms

Generalizing Li, Dong & Wang's work [LDW17], we exhibit some highest-degree terms whose coefficients are known to have a very particular structure. In order to find such monomials, we study the first two rounds of Ascon.

In the nonce-misuse setting and after one round, monomials of degree 1 (in public variables) have different coefficients depending on the row they appear on: either $a_i \oplus 1$, 1, e_i or a_i (see Table 2). After the first linear layer L_1, the situation is similar because p_L is applied row-wise. Proposition 5 immediately follows.

Proposition 5. *Let (i_0, i_1) be a fixed pair of indices. Let j be an element of the set {0, 1, 3, 4}. Let us assume that all $x_{i_0}x_{i_1}$ present in the ANF after the second S-box layer S_2 are obtained through multiplications of some monomial x_{i_1} present in the ANF after one round by some monomial x_{i_0} coming* only from *Row j. Then, all the coefficients of monomials $x_{i_0}x_{i_1}$ present in the ANF after 2 rounds share a common divisor β_{j,i_0}: $\beta_{0,i_0} = a_{i_0} \oplus 1$, $\beta_{1,i_0} = 1$, $\beta_{3,i_0} = e_{i_0}$ or $\beta_{4,i_0} = a_{i_0}$.*

The situation for highest-degree terms being completely cyclic in this setting (see Proposition 4), we fix $i_0 = 0$, *i.e.*, x_0 plays the role of x_{i_0} in Proposition 5. Then, x_0 is referred as the *primary* variable.[4] We now split the 63 remaining indices into five disjoint sets $\mathcal{S}_{a+1}, \mathcal{S}_a, \mathcal{S}_e, \mathcal{S}_0, \mathcal{S}'$:

1. $\mathcal{S}_{a+1}$ contains the indices i of the variables x_i that are *only* multiplied by some monomial x_0 which was present on *Row 0* after L_1.
2. $\mathcal{S}_a$ contains the indices i of the variables x_i that are *only* multiplied by some monomial x_0 which was present on *Row 4* after L_1.
3. $\mathcal{S}_e$ contains the indices i of the variables x_i that are *only* multiplied by some monomial x_0 which was present on *Row 3* after L_1.
4. $\mathcal{S}_0$ contains the indices of the variables x_i that are *never multiplied* by x_0 during S_2.
5. $\mathcal{S}'$ contains the remaining indices: indices of the variables either multiplied by some monomial x_0 coming only from Row 1, or by some x_0 coming from multiple rows.

The sets $\mathcal{S}_{a+1}, \mathcal{S}_a, \mathcal{S}_e, \mathcal{S}_0, \mathcal{S}'$ are given in Table 3. For any other choice of the primary variable, the indices in each set are shifted (modulo 64) by the index of the primary variable.

[3]The addition is computed modulo 64.

[4]Another choice of primary variable is possible and corresponds to a mere shift of all the indices.

Table 3: The sets $\mathcal{S}_{a+1}, \mathcal{S}_a, \mathcal{S}_e, \mathcal{S}_0, \mathcal{S}'$.

Set	Size	Indices (relative to the index of the primary variable)
$\mathcal{S}_{a+1}$	6	9, 12, 18, 19, 21, 28
$\mathcal{S}_a$	5	7, 24, 41, 43, 52
$\mathcal{S}_e$	5	17, 35, 40, 46, 55
$\mathcal{S}_0$	22	1, 4, 5, 6, 8, 14, 15, 16, 26, 27, 30, 34, 37, 38, 48, 49, 50, 56, 58, 59, 60, 63
$\mathcal{S}'$	25	2, 3, 10, 11, 13, 20, 22, 23, 25, 29, 31, 32, 33, 36, 39, 42, 44, 45, 47, 51, 53, 54, 57, 61, 62

Thanks to Proposition 5, we recover some valuable information about the coefficients of some highest-degree terms after two rounds.

Corollary 3. *Let $i \in \{1, \dots, 63\}$.*

1. *If $i \in \mathcal{S}_{a+1}$, then coefficients of all x_0x_i after S_2 share $a_0 \oplus 1$ as a common divisor.*
2. *If $i \in \mathcal{S}_a$, then coefficients of all x_0x_i after S_2 share a_0 as a common divisor.*
3. *If $i \in \mathcal{S}_e$, then coefficients of all x_0x_i after S_2 share e_0 as a common divisor.*
4. *If $i \in \mathcal{S}_0$, then x_0x_i never appears in the output of S_2.*

From there, we exhibit two classes of highest-degree monomials after 6 rounds whose coefficients have a very particular structure. In order to build cubes of dimension 32, 31 indices have to be chosen, as we already (arbitrarily) selected x_0.

Proposition 6. *Let $\mathcal{S}$ be a subset of $\mathcal{S}_{a+1} \cup \mathcal{S}_e \cup \mathcal{S}_0$ of size 31. Let v be the 64-bit vector defined by* **Supp**$(v) = \{0\} \cup \mathcal{S}$*. Let $\alpha_{v,i}$ be the coefficient associated to the monomial x^v in the ith output coordinate after 6 rounds, $i \in \{0, \dots, 63\}$. Then, $\alpha_{v,i}$ can be written as $\alpha_{v,i} = (a_0 \oplus 1)\gamma_{v,i,a} + e_0\gamma_{v,i,e}$.*

Proof. According to Corollary 1, a coefficient associated to a degree-32 monomial after 6 rounds can be seen as a sum of products of coefficients of degree-2 monomials after 2 rounds. For a coefficient associated to x^v, in each of these products, exactly one coefficient of degree 2 corresponds to a coefficient associated to a monomial x_0x_i with $i \in \mathcal{S}$ which was present after 2 rounds. But according to Corollary 3, all the coefficients of x_0x_i with $i \in \mathcal{S}$ are either 0 or divisible by $a_0 + 1$ or e_0. The result follows immediately. □

By changing the set $\mathcal{S}$ in Proposition 6, a similar result is obtained for another class of coefficients, with a different decomposition.

Proposition 7. *Let $\mathcal{S}$ be a subset of $\mathcal{S}_a \cup \mathcal{S}_e \cup \mathcal{S}_0$ of size 31. Let w be the 64-bit vector defined by* **Supp**$(w) = \{0\} \cup \mathcal{S}$*. Let $\alpha_{w,i}$ be the coefficient associated to the monomial x^w in the ith output coordinate after 6 rounds, $i \in \{0, \dots, 63\}$. Then, $\alpha_{w,i}$ can be written as $\alpha_{w,i} = a_0\varphi_{w,i,a} + e_0\varphi_{w,i,e}$.*

Based on the previous observations, our cube attack focuses on two particular monomials of degree 32, x^v and x^w, chosen according to Propositions 6 and 7 respectively. In contrast with previous works, these cubes are not chosen based on some heuristics. Instead, this choice follows the simple rationale provided by Propositions 6 and 7, and among all possible choices, we selected the whole sets $\mathcal{S}_e$ and $\mathcal{S}_0$, and the 4 smallest indices of $\mathcal{S}_{a+1}$ for x^u (resp. of $\mathcal{S}_a$ for x^w), as shown in Table 4.

Table 4: Supports of v and w.

	Primary index	$\mathcal{S}_{a+1}$	$\mathcal{S}_a$	$\mathcal{S}_e$	$\mathcal{S}_0$	$\mathcal{S}'$
Supp(v)	0	9, 12, 18, 19	-	17, 35, 40, 46, 55	1, 4, 5, 6, 8, 14, 15, 16, 26, 27, 30, 34, 37, 38, 48, 49, 50, 56, 58, 59, 60, 63	-
Supp(w)	0	-	7, 24, 41, 43	17, 35, 40, 46, 55	1, 4, 5, 6, 8, 14, 15, 16, 26, 27, 30, 34, 37, 38, 48, 49, 50, 56, 58, 59, 60, 63	-

Remark 3. With the same reasoning as in Propositions 6 and 7, we cannot find a monomial of degree 32 and guarantee that all the coefficients share a common divisor. Indeed $|\mathcal{S}_0 \cup \mathcal{S}_\alpha| < 31$ for any $\alpha \in \{a+1, a, e\}$. Interestingly, this can be done when looking at fewer rounds, or in the case of round-reduced initializations. In [LDW17], Li, Dong & Wang, present attacks against 5-round and 6-round initializations which use some monomials, whose coefficients have common linear factors *which come from such a choice of variables*. In the supplementary material provided with our work[5], those choices are explained thanks to the framework we introduced, using a SageMath [The20] script.

5 Capacity-Recovery Attack against the Full Encryption

We present an *adaptative chosen-plaintext attack* against nonce-misused Ascon, based on the full recovery of the inner-state Σ_{E}. This recovery is made of three steps which all are based on the same principle. At each step, we target some chosen monomials. From the particular form of their coefficients, as well as the value of the corresponding cube-sum, information about the capacity can be easily deduced. The attack can be decomposed as follows:

1. The first step (Section 5.1) recovers all e_i and (in average) half of the bits a_i, from the values of the cube-sums corresponding to the two monomials x^v and x^w defined in Table 4, and to their rotated versions. These bits are deduced from the general form of the coefficients of these two monomials, as exhibited in Propositions 6 and 7, and their expressions are not explicitly computed.

2. The second step (Section 5.2) recovers the remaining a_i by a classical cube-attack targeting some other highest-degree monomials. Indeed, the knowledge of the capacity bits recovered at Step 1 enables us to easily compute the exact expressions of their coefficients.

3. The third step (Section 5.3) recovers most of the bits b_i and c_i by targeting some sub-leading monomials, whose coefficients are sparse polynomials of degree at most 2 in these unknowns. The few remaining bits b_i and c_i are eventually recovered by an exhaustive search.

5.1 First Step: Recovering Most of the Bits a_i and e_i

In the first step, we recover all the values of the bits e_i and most of the ones of the bits a_i. To do so, we mount a conditional cube attack using x^v and x^w defined in Table 4, and their respective 63 siblings $x^{v \ggg k}, x^{w \ggg k}$ obtained by rotating the vectors v and w by k positions, $k \in \{1, \dots, 63\}$.

[5]https://github.com/baudrin-j/practical_cube_attack_against_nonce_misused_ascon

In Ascon, a cube-sum is a vector of size 64 corresponding to the 64 cube-sums associated to a fixed monomial, for all the 64 output coordinates. In the following algorithms, the function `CubeSumVector`, which takes a monomial x^u as input, refers to the computation of such a vector by summing the outputs corresponding to all public variables $x \preccurlyeq u$ (see Proposition 1).

Because of Proposition 6, we observe that, if the cube-sum vector associated to x^v is not the zero vector, then $a_0 \oplus 1 = 1$ or $e_0 = 1$. However *in theory*, without further studying $\gamma_{v,i,a}, \gamma_{v,i,e}$ for all $i \in \{0, \dots, 63\}$, we can only use the aforementioned property. But, *in practice*, our experimental results described in the following paragraph, show that, when $a_0 \oplus 1 = 1$ or $e_0 = 1$, the cube-sum vector is (almost) never all-zero. This means that a "practical reciprocal" can be used and thus enables to recover information about the capacity more often than with the theoretical argument only. The same kind of arguments can be used with x^w, and thus Assumptions 1 and 2 can be stated.

Assumption 1. *Let us consider the nonce-misuse scenario. Let us assume that the cube-sum associated to the monomial x^v (see Table 4) is the zero vector. Then, the guess ($a_0 \oplus 1 = 0$ and $e_0 = 0$) is wrong with a negligible probability.*

Assumption 2. *Let us consider the nonce-misuse scenario. Let us assume that the cube-sum associated to the monomial x^w (see Table 4) is the zero vector. Then, the guess ($a_0 = 0$ and $e_0 = 0$) is wrong with a negligible probability.*

The index of the considered bits is subject to the choice of the primary variable, but according to Proposition 4, everything remains identical if we choose another primary variable. So, under Assumptions 1 and 2 (and their respective 63 shifted versions), it is possible to recover all the bits e_i and, in average, half of the bits a_i by following Algorithm 1.

Algorithm 1 Step 1: v and w are defined in Table 4.

```
Output: $e_i$ for all $i \in \{0, \dots, 63\}$ and $a_i$ for some $i \in \{0, \dots, 63\}$
  for all $i \in \{0, \dots, 63\}$ do
    $a_i \leftarrow -1, e_i \leftarrow -1$                    ▷ Initialize all variables.
  end for

  for all $i \in \{0, \dots, 63\}$ do
    $Z_v \leftarrow$ CubeSumVector($x^{v \ggg i}$)
    if $Z_v = (0, \cdots, 0)$ then
      $a_i \leftarrow 1, e_i \leftarrow 0$                      ▷ Assumption 1
    else
      $Z_w \leftarrow$ CubeSumVector($x^{w \ggg i}$)
      if $Z_w = (0, \cdots, 0)$ then
        $a_i \leftarrow 0, e_i \leftarrow 0$                    ▷ Assumption 2
      else
        $e_i \leftarrow 1$                                      ▷ No assumption
      end if
    end if
  end for
```

The cost of this first step is thus upper-bounded by $64 \times 2 = 128$ cube-sums of dimension 32. In other words, time and data costs are upper-bounded by $128 \times 2^{32} = 2^{39}$, while the memory cost is negligible. The worst case happens when $\textbf{Supp}(e) = \{0, \dots, 63\}$; the best case when $\textbf{Supp}(e) = \emptyset$.

Underpinning our Assumptions. We now provide the experimental results which led us to state Assumptions 1 and 2. Even if there might be other choices for x^v and x^w following Propositions 6 and 7, we do not expect them to behave in a drastically different manner.

Observation 1. *Let us consider the following experiment: given a random capacity, compute the cube-sum vector associated to the monomial x^v. If the cube-sum vector is all-zero, then we guess that $a_0 \oplus 1 = 0$ and $e_0 = 0$. Otherwise, no guess is made. Our experiment gave the following results: 4000 capacities were tested. A guess was made about* 25 % *of the time (1037/4000) and* 100 % *of the guesses were actually right.*

Observation 1 suggests that it is possible to detect each time that $a_0 \oplus 1 = 0$ and $e_0 = 0$. But, more importantly, it raises no false positive under Assumption 1. Observation 2 gives a similar result for the monomial x^w.

Observation 2. *Let us consider the following experiment: given a random capacity, compute the cube-sum vector associated to the monomial x^w . If the cube-sum vector is all-zero, then we guess that $a_0 = 0$ and $e_0 = 0$. Otherwise, no guess is made. Our experiment gave the following results: 4000 capacities were tested. A guess was made about* 25 % *of the time (1002/4000) and* 100 % *of the guesses were actually right.*

Our experiments show that an all-zero cube-sum vector has a very low probability of happening if $a_0 = 0$ or $e_0 = 1$ (resp. $a_0 = 1$ or $e_0 = 1$ for x^w). Indeed, when (a_0, e_0) is not equal to the targeted value, the distributions of the Hamming weights of the vectors formed by the 64 cube sums behave as the distribution of the Hamming weights of random vectors, as depicted on Figure 6: they seem to follow a binomial distribution with parameters $n = 64$, $p = 0.5$. In particular, they have an average around $64 \times 0.5 = 32$ and a low variance, indicating that the cube-sum vector should not be able to vanish when not expected. Moreover, each bit of the cube-sum vector is almost uniformly distributed, as shown on Figure 7. As the capacities were picked uniformly at random for our experiments, we expect this study to be quite representative of the overall situation, and from this, Assumptions 1 and 2 are considered to be valid.

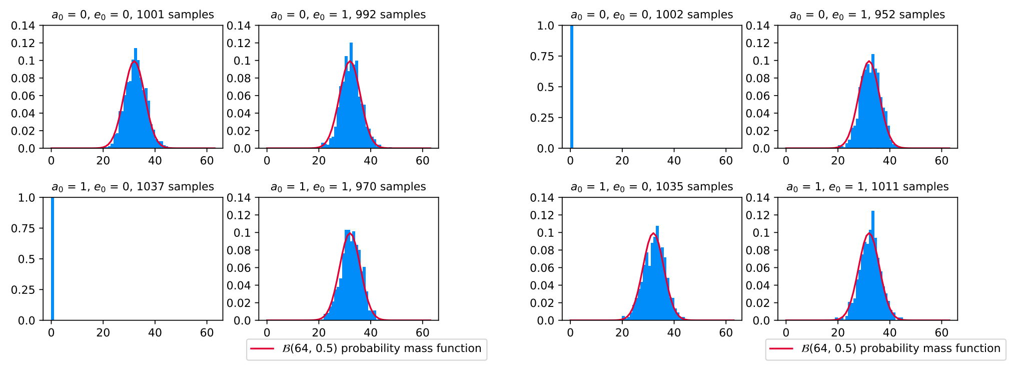

Figure 6: Distribution of the Hamming weight of cube-sum vectors for x^v (left) and x^w (right) depending on the values of (a_0, e_0), for 4000 capacities chosen uniformly at random.

5.2 Second Step: Recovering the Remaining a_i

Step 2 consists in recovering some of the not-yet-recovered values of a_i. It iteratively updates the set $L := \{i \mid a_i \text{ is still unknown}\}$, starting from $L = \mathbf{Supp}(e)$. To do so, we mount a cube-like attack by targeting some other monomials of degree 32. The choice of

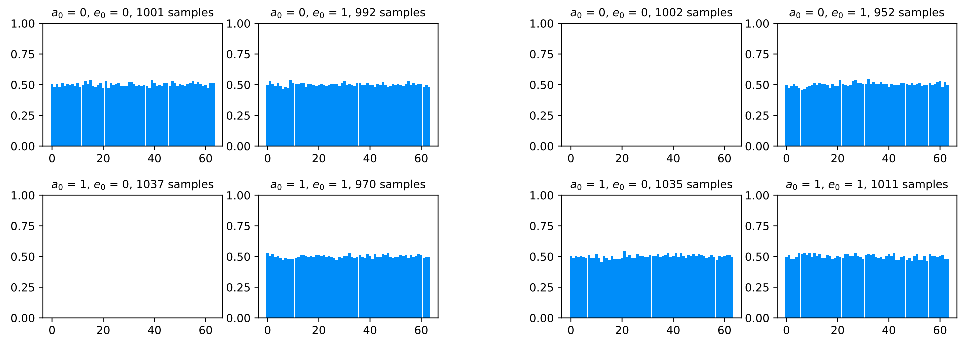

Figure 7: Ratio of 1 values taken by $\alpha_{v,i}$ (left) and $\alpha_{w,i}$ (right) for each output coordinate ($i \in \{0, \ldots, 63\}$) and for 4000 capacities chosen uniformly at random, depending on (a_0, e_0).

monomials is done adaptively and depends on the already-recovered bits of the capacity: at least one of the indices of the variables in the chosen monomials needs to be the index of an unknown a_i. The cube-sum corresponding to such a monomial x^u then leads to a polynomial system whose unknowns are some of the a_i with $i \in L \cap \textbf{Supp}(u)$. When this set of indices is not too large, typically of size at most 5, the system can be easily solved and the values of the corresponding a_i can be recovered. This procedure is described in Algorithm 2.

Algorithm 2 Overview of Step 2

```
Input: $L$, the set of indices $i$ such that $a_i$ is still unknown after Step 1.
    $A, E$, sets of index-value pairs $(i, v)$ corresponding to the recovered bits $a_i$ (resp. $e_i$)
    during Step 1.
Output: $a_i$ for most $i$ in $L$
  while L $\neq \emptyset$ do
    Choose $u \in \mathbb{F}_2^{64}$ such that $\textbf{wt}(u) = 32$, $L \cap \textbf{Supp}(u) \neq \emptyset$ and preferably
    $|L \cap \textbf{Supp}(u)| \leq 5$.
    $P \leftarrow$ ComputeCoefficients$(u, A, E)$ ▷ $P$, polynomial expressions of 64 coefficients.
    $V \leftarrow$ CubeSumVector$(x^u)$
    $S \leftarrow$ SolvePolynomialSystem$(P, V)$      ▷ $S$, (possibly empty) set of index-value
    $A \leftarrow A \cup S$                                    pairs of recovered values.
    $L$.remove$(\{i, (i, v) \in S\})$
  end while
```

More precisely, the `ComputeCoefficients` procedure in Algorithm 2 consists in computing the *exact* polynomial expressions of the coefficients of the highest-degree monomials dividing x^u, round after round. This can be done since most of the unknowns of a and e can be replaced by their actual values recovered during Step 1, while this was prohibitively expensive before[6]. This can be considered as an implemented version of the *Partial Polynomial Multiplication* method introduced by Rohit *et al.* [RHSS21].

Obviously, the choice of u, especially the size $|L \cap \textbf{Supp}(u)|$, affects the cost of solving the system. Therefore, there are some trade-offs between time and memory complexity and data complexity: we may choose to solve many easily-computed low-degree systems, or fewer high-degree systems which are also harder to build. In our experiments, we tried to limit the number of unknowns variables to 4 or 5. In these cases, the computation of

[6]During Step 1, *conditional cubes* based on *partial knowledge* of the expressions were used instead.

a coefficient is fast (a few seconds) and does not require too much RAM: a 16-Go RAM laptop was used for this step and it never ran out of memory. The systems are also quickly solved with a SAT solver (Cryptominisat [SNC09]) and can even be solved without using (and thus computing) all of the 64 coefficients. At the beginning, $L = \mathbf{Supp}(e)$, so finding u such that, $\mathbf{wt}(u) = 32$, and $|L \cap \mathbf{Supp}(u)| \leq 5$ might be impossible, depending on $\mathbf{wt}(e)$. However, for more than 91.5% (resp. 99.2%) of the possible values of e, the number of unknowns in the coefficients can be limited to 5 (resp. 9).

Overall, the online cost of Step 2 is less than 64 cube-sum computations of cubes of dimension 32. The time complexity of building and solving the systems is harder to predict. However, as an adversary can usually choose some easily-built and easily-solved highly-over-determined systems (64 equations in at most 5 to 9 unknowns), these costs remains negligible compared to the time complexity of the cube-sum computations.

Finally, the very last bits of a can be harder to recover because only constant coefficients are found. Indeed, each product described in Corollary 1 of size 32 is very likely to vanish. In that case, they can remain unknown for now: only a few values of a_i with $i \in \mathbf{Supp}(e)$ are necessary for the next stage to work properly.

5.3 Third Step: Recovering Most of the Bits b_i and c_i

Step 3 consists in recovering bits b_i and c_i for all i, while d_i can then be computed as $d_i = e_i \oplus c_i \oplus 1$. To do so, we mount a cube-like attack by targeting *sub-leading monomials*, that is, monomials of degree 31. This stage is also *adaptative* and depends on the capacity.

Because all e_i and most a_i bits have been recovered, almost all coefficients of monomials of degree 1 after L_1 can be considered constant. Corollary 2 can thus be greatly simplified.

Corollary 4. *Let u be a 64-bit vector such that $\mathbf{wt}(u) = 2^{r-1} - 1$, and α_u be the coefficient of x^u in any output coordinate after Round r. Let us assume that a_i and e_i are known for all i, $i \in \mathbf{Supp}(u)$. Then, α_u can be viewed as a sum whose terms have one of the following forms:*

- *a binary constant, if the term corresponds to a former product of 2^{r-1} coefficients of terms of degree 1, or*
- *a quadratic polynomial with monomials of the form b_i, c_i, b_ic_i for all $i \in \{0, \dots, 63\}$ (and possibly with some monomials of the form a_i, a_ib_i, a_ic_i for the few values of i for which a_i is still unknown) if the term corresponds to a former product of one constant term and $2^{r-1} - 1$ coefficients of terms of degree 1 after the second constant addition.*

Proof. The second case comes from the expression of the constant terms after S_1(see Table 2), once d_i is written in terms of variable c_i and the known value of e_i. They are the only variables influencing the constant terms after the second constant addition. □

Selecting such a sub-leading monomial then leads to sparse coefficients of degree at most 2. Indeed, if r ($r \geq 0$) values of a_i remain unknown before Step 3, the coefficients will depend on at most $128 + r$ linear terms and $64 + 2r$ quadratic terms.

However, we need to select the sub-leading monomial in a way which avoids the situation where all the corresponding coefficients after six rounds vanish. We then use the following observation.

Proposition 8. *Let us consider the nonce-misuse scenario. Let x^u be a monomial of degree 16 in public variables. Let α_u be a coefficient of x^u in any coordinate after 5 rounds. Then, each product (Corollary 1) appearing in the algebraic expression of α_u is divisible by at least one variable e_i, with $i \in \mathbf{Supp}(u)$.*

Proof. This can be proved by keeping track, round after round, of the coordinates in which all coefficients associated to highest-degree monomials dividing x^u, have the expected property. Bounds on the number of e_i appearing in each product are given in Table 5. □

Table 5: Lower and upper bounds on the number of e_i variables appearing in each product (Corollary 1) of the coefficient of any 2^{r-1}-degree monomial at Round r.

Round	1	2	3	4	5	6
Row r_0	0	0	1-2	1-4	**1**-7	2-13
Row r_1	0	1	1-2	0-3	**1**-6	3-15
Row r_2	x	1	0-1	1-2	**3**-7	3-15
Row r_3	1	0-1	0-1	1-3	**2**-8	2-15
Row r_4	0	0	1	2-4	**1**-7	2-13

Corollary 5. *Let us consider the nonce-misuse scenario. Let x^u be a monomial of degree 31 in public variables. Let us suppose that $e_i = 0$ for all $i \in \mathbf{Supp}(u)$. Then, for any output coordinate after 6 rounds, the coefficient α_u of x^u is the null polynomial.*

Apart from the selection process of the monomials, Step 2 and Step 3 follow the same process, already presented in Algorithm 2. However, contrary to Step 2, the set $\mathbf{Supp}(u) \cap \mathbf{Supp}(e)$ does not influence the degree anymore: it now only affects the number of variables and the sparsity of the resulting system of 64 equations. Choosing a small set $\mathbf{Supp}(u) \cap \mathbf{Supp}(e)$ makes the computation of the expressions of the coefficients faster because more products vanish (see Corollary 2). On the other hand, a larger set allows the recovery of more unknowns at one stroke. We preferred keeping $|\mathbf{Supp}(u) \cap \mathbf{Supp}(e)|$ around 4 or 5. In that case, the expressions of the coefficients are easily computed: it took about 5 to 25 minutes on two AMD EPIC 7352 (24 cores, 2.3GHz) to compute in parallel the 64 expressions of the coefficients of a chosen monomial. Moreover, by considering the number of terms after the first rounds, we avoided the longest computations. Indeed, this number is a good indicator of the amount of remaining work and thus of, whether it is worth continuing or not.

Overall, at least 128 equations are needed to recover the 128 bits of b and c, so at least two sets of 64 coefficients and two cube-sums of dimension 31 have to be computed.[7] Using 3 random monomials is *often* enough to recover all the unknown bits except at most 10 of them and we expect that it will almost always cost at most 10 cube-sums.

5.4 Finalizing the Recovery

After the three steps, a small number of unknown capacity bits is expected to remain. They can be recovered through an exhaustive search. In the end, the complexity of the attack is dominated by the cost of the cube-sums of degree-32 monomials, that is, by $(128+64)2^{32} \approx 2^{7.6+32} < 2^{40}$ both in time and data.

It is hard to give an explicit formula for the time complexity of the adaptative phases during Steps 2 and 3. However, from our experiments, they can be effectively mounted on a personal computer and a single cluster node for Step 2 and 3 respectively. The entire process did not last more than a few hours on two AMD EPIC 7352. We provide as

[7]Contrary to the case of coefficients of degree-32 monomials, the dependency of coefficients of degree-31 monomials is not only limited to variables b_i, c_i with $i \in \mathbf{Supp}(u)$. This is because, through the first linear layer L_1, constant terms are shuffled, contrary to terms of degree 1 which are only copied in other coordinates.

supplementary material[8] all the files used to effectively mount the attack, as well as some details about our implementation choices (see Appendix B).

5.5 Comparing our Results with [CHK22]

Another conditional cube attack against nonce-misused Ascon has been exhibited in a concurrent work by Chang, Hong & Kang [CHK22]. Both our attack and theirs were proposed independently. They both use conditional cubes to first recover the 128 bits of a and e. Similarities (pointed out in Table 6) appear between our cube x^w and their `Pattern-A`: 27 out of the 32 variables involved in the cube are identical.

It is due to a common desire to *conditionally* study the (dis)appearance of quadratic monomials involving a_0 in their coefficients after the second round.

Table 6: Comparative study of our attack with [CHK22].

	Our attack		[CHK22]
Recovery of a and e	Conditional cube	$=$	Conditional cube
	$\mathcal{S}_a \cup \mathcal{S}_0$	$=$	$\{v_1, \cdots, v_{27}\}$
	primary variable	$=$	*conditional cube variable*
	Supp$(w) =$ $\{0\} \cup \mathcal{S}_0 \cup (\mathcal{S}_a \setminus \{52\}) \cup \mathcal{S}_e$	$\approx$	`Pattern-A` $=$ $\{0\} \cup \mathcal{S}_0 \cup \mathcal{S}_a \cup \{2, 9, 12, 18\}$
	2^{39} in time and data	$\approx$	$2^{44.8}$ in time and data
Recovery of b and c	*Ad hoc* cube-like attack	$\neq$	Exhaustive search
	$\approx 2^{38}$ in time and data	$\neq$	2^{128} in time

$=, \approx$ stand for exact and partial correspondence, $\neq$ stands for no correspondence.

But to define a monomial of degree 32, 5 additional variables need to be chosen. From there, the directions taken by both parties greatly differ. They can be compared by classifying the quadratic monomials in the ANF of 2-round Ascon in the nonce-misused scenario (which is easily computed with SageMath [The20]). This classification is presented in Figure 8.

Under the assumption that $a_0 = 0$, it can be observed that 27 monomials do not appear in the ANF of 2-round Ascon: Chang, Hong & Kang selected all corresponding variables in their pattern, while we only selected 26[9]. The remaining $63 - 27 = 36$ monomials split into subsets depending on whether or not all $x_0 x_i$ could disappear from the ANF if other conditions were added to condition $a_0 = 0$. Among them, $12 + 7 = 19$ monomials (represented in the bottom right-hand corner of Figure 8) can disappear from the ANF if a single linear condition (depending on i) is added to $a_0 = 0$:

- 7 monomials are such that the GCD of the coefficients of $x_0 x_i$ is divisible by e_0. Their indices form the set $\mathcal{S}_e \cup \{10, 13\}$. We chose the remaining $32 - (1 + 26) = 5$ variables among $\mathcal{S}_e$.
- The 12 remaining ones have 12 distinct linear GCDs *independent from* e_0; they form the set $\{w_1, \cdots, w_{12}\}$ [CHK22] in which the authors chose the remaining $32 - (1 + 27) = 4$ cube variables.

[8] `https://github.com/baudrin-j/practical_cube_attack_against_nonce_misused_ascon`

[9] As explained in Section 4.2, this is only because the variables of $\mathcal{S}_a$ were the last ones of the 32 variables that we selected, so all variables of $\mathcal{S}_a$ *except one* were selected.

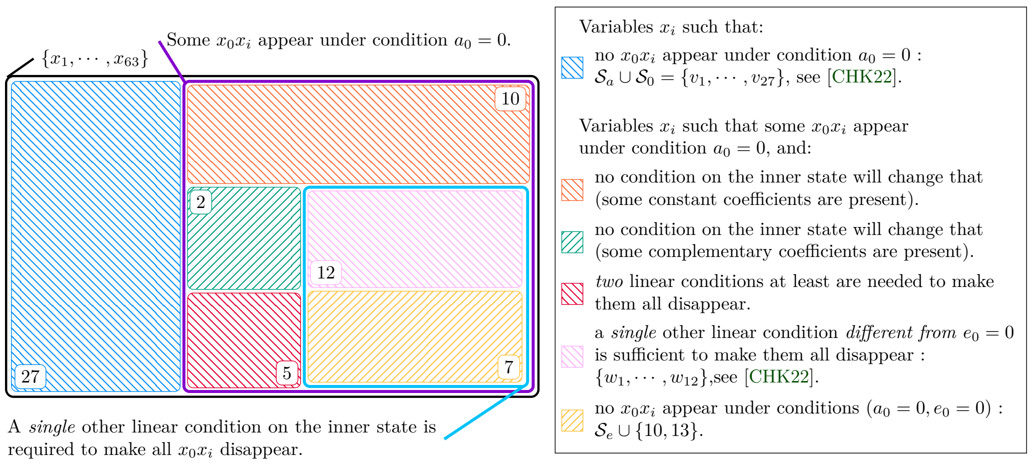
Some x_0x_i appear under condition $a_0 = 0$.
$\{x_1, \cdots, x_{63}\}$
10
2
12
27
5
7
A *single* other linear condition on the inner state is required to make all x_0x_i disappear.
Variables x_i such that:
no x_0x_i appear under condition $a_0 = 0$: $\mathcal{S}_a \cup \mathcal{S}_0 = \{v_1, \cdots, v_{27}\}$, see [CHK22].
Variables x_i such that some x_0x_i appear under condition $a_0 = 0$, and:
no condition on the inner state will change that (some constant coefficients are present).
no condition on the inner state will change that (some complementary coefficients are present).
two linear conditions at least are needed to make them all disappear.
a *single* other linear condition *different from* $e_0 = 0$ is sufficient to make them all disappear : $\{w_1, \cdots, w_{12}\}$,see [CHK22].
no x_0x_i appear under conditions $(a_0 = 0, e_0 = 0)$: $\mathcal{S}_e \cup \{10, 13\}$.

Figure 8: Overview of the quadratic monomials x_0x_i, $i \in \{1, \ldots, 63\}$ after the second round.

Our choice corresponds to variables i such that all terms x_0x_i vanish when $e_0 = 0$ and $a_0 = 0$ (see Prop. 7). The choice from [CHK22] is different since *a linear condition per variable* is needed in addition to condition $a_0 = 0$; hence $4 + 1 = 5$ conditions. This is the reason why their attack does not work for any value of the state Σ_E and needs to be repeated for 32 triples of key, nonce and associated data in average, until the state satisfies the required conditions. Moreover, it is worth noting that minimizing the number of conditions per cube has several advantages.

1. It enables a simpler study of the "practical reciprocal": with k conditions, 2^k cases have to be studied to experimentally verify the necessary assumptions (see Figures 6 and 7 for an example when $k = 2$).

2. Recovering information when the cube-sum is *not the all-zero vector* is also easier. In that case, the adversary recovers the value of the OR of all the negated conditions. The smaller the number of conditions is, the easier it is to recover information from multiple recovered OR; see for example the last *else* case in Algorithm 1.

3. Finally, the fewer conditions, the easier to partition the set of all inner states into disjoint subsets, thus enabling *independent* recoveries of bits; see for example x^v (or x^w) and its shifted values.

The recoveries of the last 128 bits are also entirely different, as pointed out in Table 6.

6 Counter-Measures

Identifying the properties that make our attack possible is of great interest, especially for designers. The first step of our attack plays the most crucial role since the other two steps seem infeasible without the initial recovery of 96 bits in average. We thus focus on counter-measures that avoid the use of cubes similar to those involved in Step 1.

As in all previous cube attacks against Ascon, the targeted monomials in the first step are chosen to have the highest possible degree. The reasons of this choice are at least two-fold. First, it corresponds to the coefficients which *a priori* depend on the smallest possible number of unknown variables: without any further study, they seem less intricate than any others. Moreover, because of the quadratic S-box, together with a (reduced) number of rounds which is quite small, their values are still quite cheap to recover.

Obviously, increasing the number of rounds during encryption prevents cube attacks. In the nonce-misuse setting, because there are 64 public variables, *only a single highest-degree term* could be exploited after seven rounds. Moreover, all coefficients of this degree-64 monomial would depend on up to twice as many unknown variables as in our attack, and the data limitation of 2^{64} would be reached. However, this trivial counter-measure comes with a major drawback, especially in the case of a lightweight cipher: it increases the costs of encryption and decryption and lowers the throughput.

A more-interesting counter-measure could be to change the outer part of the internal state (the rate) during encryption: instead of inserting the plaintext within the first word of the state, we could consider XORing it with any of the four other words. This would not change anything regarding the performance of Ascon but it would however lead to different algebraic properties. Indeed, contrary to its affine-equivalent form χ [BDPV11b], the S-box of Ascon is not rotation-invariant, so changing the position of the outer state changes the general form of the outputs. We studied the four other possibilities and tried to find some conditional cubes by following the same reasoning we used for the genuine outer part of the state. As a result, any of the four other choices is achieving a better resistance against our method than the current setting. Compared to the genuine setting, a single choice leads to a lower average number of recovered bits, while the three others do not enable to find any conditional cubes of dimension 32 as we did in our attack.

The main observations are summarized in Table 8 (Appendix C). The slow diffusion of the public variables through the first rounds, and more specifically the low number of distinct quadratic monomials enables to build conditional cubes as we did. Inserting public variables in different columns while the S-boxes are applied column-wise limits the number of quadratic monomials after the first rounds. For the genuine setting, this phenomenon is accentuated by the absence of all public variables on the third row after L_1. It also occurs when the outer part of the state corresponds to the third 64-bit word of the state. When initialization is targeted, as done in [LDW17, DEMS15, RHSS21, RS21], the same observations can be made when inserting public variables (corresponding to the nonce) on the third row and keeping the fourth initial row all-zero, or by inputting the same variables on both rows.

It seems that the sparsity of some of the coordinates of the S-box is the main cause: another S-box might achieve better resistance against (conditional) cube attacks, but at the probable cost of an increased number of gates. The relevance of changing the S-box goes beyond the scope of this work, but its study remains an interesting challenge.

7 Conclusion

To the best of our knowledge, this work presents the first state-recovery attack on the full 6-round encryption of Ascon, under the assumption that a key-nonce pair is reused many times. Besides this particular result, we introduce a new technique for searching for conditional cubes in Ascon. It is based on the splitting of the set of public variables, depending on the coefficients of quadratic monomials after two rounds. This method explains the observations in [LDW17] on the round-reduced initialization of Ascon. It also generalizes the general form of the coefficients that can be targeted in a conditional attack compared to the previous attacks presented in [HWX+17] and [LDW17] against Keccak and Ascon.

The first step of our approach enables the recovery of 64 to 128 bits of the internal state. Thanks to this knowledge, the computation of the previously-inaccessible part of the ANF is now possible. The recovery of the remaining bits thus follows. Overall, our adaptative chosen-plaintext attack can recover the full internal state, that is, 256 unknown bits, with an online time and data complexity lower than 2^{40}. Some trade-offs between time, memory and data are also presented. Our attack is practical: it has been implemented and it successfully returns the secret capacity bits with the expected complexity. Finally, we mentioned possible counter-measures against this kind of recovery based on cube-attacks, which might be relevant in the context of the current standardization process launched by the NIST.

It is already known that the nonce-misuse scenario enables the recovery of any plaintext at the cost of an adapted query for each block that needs to be decrypted [VV18]. But, in our attack, once the inner-state has been recovered, an adversary can recover all the following plaintexts from the corresponding ciphertexts, without any further interaction with the encryption oracle. Nevertheless, this result does not go against the claims made by the designers of Ascon [DEMS19], nor does it unsettle the confidence gained through the previous studies, as it is clearly stipulated in the specifications that the nonce must not be "repeated for two encryptions under the same key". However, we believe that such attacks are interesting because implementation errors cannot be completely ruled out. Moreover, we think that this work provides further understanding about the link between conditional cubes and the Algebraic Normal Form, especially in the case of Ascon.

Acknowledgments

We would like to thank Chang, Hong and Kang for the open and useful discussions about their independent work [CHK22]. They analyzed Ascon and obtained results that have some overlap with ours. Jules Baudrin is funded by ANR (French National Research Agency) grant ANR-20-CE48-0017 (SELECT). The work of Léo Perrin is supported by the European Research Council (ERC, grant agreement no. 101041545 "ReSCALE").

References

[BDPV11a] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Cryptographic sponge functions, 2011. https://keccak.team/sponge_duplex.html.

[BDPV11b] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, 2011. https://keccak.team/keccak.html.

[BDPV12] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In Ali Miri and Serge Vaudenay, editors, *SAC 2011*, volume 7118 of *LNCS*, pages 320–337, Toronto, Ontario, Canada, August 11–12, 2012. Springer, Heidelberg, Germany.

[BN00] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In Tatsuaki Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 531–545, Kyoto, Japan, December 3–7, 2000. Springer, Heidelberg, Germany.

[CAE14] CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness, March 2014. https://competitions.cr.yp.to/caesar.html.

[CHK22] Donghoon Chang, Deukjo Hong, and Jinkeon Kang. Conditional Cube Attacks on Ascon-128 and Ascon-80pq in a Nonce-misuse Setting. Cryptology ePrint Archive, Report 2022/544, 2022. https://ia.cr/2022/544.

[DEMS] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon TikZ figures. https://ascon.iaik.tugraz.at/resources.html.

[DEMS15] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Cryptanalysis of Ascon. In Kaisa Nyberg, editor, *CT-RSA 2015*, volume 9048 of *LNCS*, pages 371–387, San Francisco, CA, USA, April 20–24, 2015. Springer, Heidelberg, Germany.

[DEMS19] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2. Technical report, National Institute of Standards and Technology, 2019. https://csrc.nist.gov/Projects/lightweight-cryptography/finalists.

[DS09] Itai Dinur and Adi Shamir. Cube attacks on tweakable black box polynomials. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 278–299, Cologne, Germany, April 26–30, 2009. Springer, Heidelberg, Germany.

[HWX$^+$17] Senyang Huang, Xiaoyun Wang, Guangwu Xu, Meiqin Wang, and Jingyuan Zhao. Conditional cube attack on reduced-round Keccak sponge function. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 259–288, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

[Jea16] Jérémy Jean. TikZ for Cryptographers. https://www.iacr.org/authors/tikz/, 2016.

[LDW17] Zheng Li, Xiaoyang Dong, and Xiaoyun Wang. Conditional cube attack on round-reduced ASCON. *IACR Trans. Symm. Cryptol.*, 2017(1):175–202, 2017.

[LZWW17] Yanbin Li, Guoyan Zhang, Wei Wang, and Meiqin Wang. Cryptanalysis of round-reduced ASCON. *Sci. China Inf. Sci.*, 60(3):38102, 2017.

[RHSS21] Raghvendra Rohit, Kai Hu, Sumanta Sarkar, and Siwei Sun. Misuse-free key-recovery and distinguishing attacks on 7-round Ascon. *IACR Trans. Symm. Cryptol.*, 2021(1):130–155, 2021.

[Rog02] Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 98–107, Washington, DC, USA, November 18–22, 2002. ACM Press.

[RS21] Raghvendra Rohit and Santanu Sarkar. Diving deep into the weak keys of round reduced Ascon. *IACR Trans. Symmetric Cryptol.*, 2021(4):74–99, 2021.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009.

[The20] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.2)*, 2020. https://www.sagemath.org.

[VV18] Serge Vaudenay and Damian Vizár. Can Caesar Beat Galois? - Robustness of CAESAR candidates against nonce reusing and high data complexity attacks. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 476–494, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany.

A Cube-like Distinguishers against Ascon

Table 7: Summary of the cube-like distinguishers against Ascon.

Attack type	Target / Scenario	Number of rounds	Data / Time	Method	Source
Nonce-respecting distinguisher	Permutation	12/12	2^{130}	Zero-sum	[DEMS15]
	Initialization	6/12	2^{33}	Cube-tester	[DEMS15]
		6/12	2^{31}		[RHSS21]
		6/12	2^{17} †		[RS21]
		7/12	2^{60}		[RHSS21]
		7/12	2^{33} †		[RS21]
Nonce-misuse distinguisher	Encryption	6/12	2^{33}	Cube-tester	[DEMS15]

† stands for "Weak-key subspace".

B About our Implementation Choices in the 2nd and 3rd Steps

As already stated, we provide along with this paper all the files used to mount the attack[10]. We would like to comment the choices we made while implementing our attack, as most of them are highly-related to the structure of Ascon.

First of all, we always omit the sixth linear layer and instead, compute all the necessary coefficients after the sixth S-box layer. Indeed, as the linear layer is invertible and applied row-wise, the cube-sum vector after the sixth S-box layer can be easily obtained from the value of the cube-sum vector after the sixth linear layer, by applying p_L^{-1} (and then its restriction to Row 0, Σ_0^{-1}) to the output cube-sum vector, as shown by the following formula.

$$\sum_{v\in C} p_L^{-1} \circ p^6(v,0) = p_L^{-1}\left(\sum_{v\in C} p^6(v,0)\right)$$

Meanwhile, even if the last two stages of our attack do not target monomials of the same degree, the main ideas used to compute the coefficients are almost the same. We here describe the main techniques we used to implement the second stage; everything being quite similar for the last stage.

1. We first build an initial state with the necessary information only: only the public variables which divide the targeted monomial are inserted as they are the only ones able to influence the targeted monomial. The remaining public variables are omitted. When the value of a bit has already been recovered (during the first stage or at the beginning of the second stage), it is inserted as a value and not as a variable in order to avoid a useless growth of the expressions round after round. For example, there is no need to keep a coefficient of the form a_0p in memory when the value of a_0 is known: we either discard it if $a_0 = 0$ or only keep p if $a_0 = 1$. This is the main reason of the avalanche effect we observed: the more public variables are already recovered, the sparser is the initial state and the easier it is to recover part of the ANF which was previously impossible to access.

[10] https://github.com/baudrin-j/practical_cube_attack_against_nonce_misused_ascon

2. Then, we go through the first four rounds by only keeping terms that may be able to influence the targeted monomial and its coefficients after 6 rounds. In other words, because of Proposition 2, only the highest-degree terms need to be stored round after round. It enables us to save some memory while also avoiding useless computations during the next rounds.

3. After the first round, we only consider the quadratic part of the S-box instead of the whole S-box. Indeed, we know from Proposition 2 that terms of highest-degree can only be obtained through a product within the S-box layer. Thus, linear terms of the S-box only diffuse terms which will never influence any highest-degree terms in the following rounds. The linear part of the S-box can thus be omitted.

4. Furthermore, we prefer storing f as a keyed Boolean function (as introduced in Section 2):

$$f = \sum_{u \in \mathbb{F}_2^{64}} \alpha_u x^u, \text{ with } \alpha_u \in \mathbb{F}_2[a_0, \cdots, e_{63}]/(a_0^2 + a_0, \cdots, e_{63}^2 + e_{63}) \, ,$$

   rather than as a polynomial of $\mathbb{F}_2[x_0, \cdots, x_{63}, a_0, \cdots, e_{63}]/(x_0^2+x_0, \cdots, x_{63}^2+x_{63}, a_0^2+a_0, \cdots, e_{63}^2 + a_{63})$. With such a representation, during the third S-box layer, the product $(a_1+1)x_0x_1 \times x_0x_2$ is discarded in one check (because $x_0x_1 \times x_0x_2 = x_0x_1x_2$ is not a highest-degree term), while it would take two checks if it was stored as $(a_1x_0x_1 + x_0x_1) \times x_0x_2$. This choice of representation enables us to save a lot of useless verifications, as the coefficients will have more and more terms as the number of rounds increases.

5. In order to go through the fifth round and the sixth S-box layer, we study the ANF of the quadratic part of the S-box (see Figure 3). It can be observed that none of the quadratic terms of the first coordinate depend on the fourth input x_3. Since we here focus on the computation of the highest-degree terms appearing in the first row after the 6-th S-box layer, we do not need to compute the two highest-degree terms appearing in the fourth row after the 5-th S-box layer.

6. Finally, the cost of the last S-box layer is not as high as one could expect. For the first rounds, a monomial of degree 2^{r-1} can be multiplied by at most $\binom{32-2^{r-1}}{2^{r-1}}$ coprime monomials of degree 2^{r-1} in order to reach the maximal degree. In the final S-box layer, $\binom{32-16}{16} = \binom{16}{16} = 1$, so given a monomial of degree 16, only a single multiplication with another monomial of degree 16 can lead to the targeted monomial of degree 32. This means that the final products of coordinates can be done without computing actual products. Rather, for a given monomial in the first coordinate, we have to check whether its "complementary" monomial is present in the second coordinate, leading to a cost linear in the number of terms of the first coordinate[11] rather than a quadratic cost.

[11]At this stage, the coordinates are stored as hash tables, so verifying the presence of a monomial can be done in constant time.

C Counter-Measure: Changing the Input Row

Table 8: A possible counter-measure: changing the public variables input row. The table provides the sizes of the sets corresponding to $\mathcal{S}_{a+1}, \mathcal{S}_a, \mathcal{S}_e$, and $\mathcal{S}_0$ given in Table 4. For instance, the first row in the table states that, for 5 indices i, the coefficients of all x_0x_i share $(a_0 + b_0 + d_0 + 1)$ as a factor.

State after initialization	Linear terms after S_1	Size of the sets	Analysis
a_0	$(a_0 + b_0 + d_0 + 1)x_0$	5	$5 + 3 + 5 + 12 < 31$ No cube as in Section 5.
x_0	$(b_0 + c_0 + 1)x_0$	3	
b_0	x_0		
c_0	x_0		
d_0	$(a_0 + d_0 + 1)x_0$	5	
Nb of variables not multiplied by x_0 after S_2		12	
a_0	$(b_0 + 1)x_0$	4	$4 + 6 + 23 > 31$. Cubes can be built as in Section 5 and should enable to detect whenever $(b_i, c_i) = (1, 0)$ (32 of the 256-bit state in avg.).
b_0	$(b_0 + c_0 + 1)x_0$	6	
x_0	x_0		
c_0	x_0		
d_0	$*$		
Nb of variables not multiplied by x_0 after S_2		23	
a_0	x_0		$3 + 4 + 5 + 12 < 31$ No cube as in Section 5.
b_0	$(b_0 + c_0 + 1)x_0$	3	
c_0	d_0x_0	4	
x_0	$(a_0 + 1)x_0$	5	
d_0	x_0		
Nb of variables not multiplied by x_0 after S_2		12	
a_0	b_0x_0	5	$5 + 4 + 5 + 5 + 12 = 31$ but b_0 and $b_0 + 1$ cannot be used at the same time. No cube as in Section 5.
b_0	x_0		
c_0	$(d_0 + 1)x_0$	4	
d_0	$(a_0 + 1)x_0$	5	
x_0	$(b_0 + 1)x_0$	5	
Nb of variables not multiplied by x_0 after S_2		12	